Knowledge Modeling - State of the Art

Vladan Devedzic

Department of Information Systems, FON - School of Business Administration
University of Belgrade
POB 52, Jove Ilica 154, 11000 Belgrade, Yugoslavia
Phone: +381-11-3950856, +381-11-4892668 Fax: +381-11-461221
Email: devedzic@galeb.etf.bg.ac.yu, devedzic@fon.fon.bg.ac.yu
URL: http://galeb.etf.bg.ac.yu/~devedzic/

Abstract. A major characteristic of developments in the broad field of Artificial Intelligence (AI) during the 1990s has been an increasing integration of AI with other disciplines. A number of other computer science fields and technologies have been used in developing intelligent systems, starting from traditional information systems and databases, to modern distributed systems and the Internet. This paper surveys knowledge modeling techniques that have received most attention in recent years among developers of intelligent systems, AI practitioners and researchers. The techniques are described from two perspectives, theoretical and practical. Hence the first part of the paper presents major theoretical and architectural concepts, design approaches, and research issues. The second part discusses several practical systems, applications, and ongoing projects that use and implement the techniques described in the first part. Finally, the paper briefly covers some of the most recent results in the fields of intelligent manufacturing systems, intelligent tutoring systems, and ontologies.

1. Introduction

There were several major research, development, and technological streams in computer science and engineering during the last decade. Some of them have had deep impact on development of intelligent systems. Together, they form a context within which modern knowledge modeling techniques should be discussed. Such streams include object-oriented software design [9], [32], layered software architectures [56], development of hybrid systems [41], multimedia systems [23], and, of course, distributed systems and the Internet [36].
Along with such a context, any discussion of knowledge modeling should also include a reference to the kinds of knowledge that can be represented in the knowledge base of an intelligent system, as well as to the basic representation and design techniques.

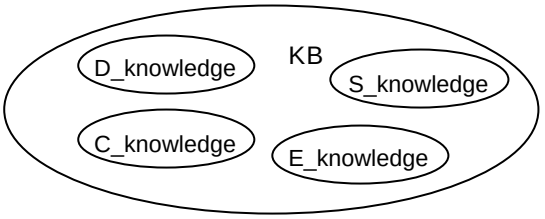

Figure 1. Knowledge base contents

Conceptually, we can think of the knowledge base as of a large, complex, aggregated object [18]. Its constituent parts can contain knowledge of different kinds. Some of them are represented in Figure 1. D_knowledge stands for *domain knowledge*, and it refers to the application domain facts, theories, and heuristics. C_knowledge stands for *control knowledge.* It describes the system's problem solving strategies and its functional model, and is more or less domain independent. E_knowledge denotes *explanatory knowledge.* It defines the contents of explanations and justifications of the system's reasoning process, as well as the way they are generated. *System knowledge* (the S_knowledge part) describes the contents and structure of the knowledge base, as well as pointers to some useful programs, which should be "at hand" during the knowledge base building process, since they can provide valuable information. Examples of such programs are various application and simulation programs, encyclopedias, etc. In some intelligent systems, system knowledge also defines user models and strategies for the system's communication with its users.
Apart from these four kinds of knowledge there can also be some other specific kinds of knowledge in the knowledge base (e.g., knowledge specific to truth maintenance, or knowledge specific to the capabilities of communication and integration with other systems).
All kinds of knowledge in the knowledge base are represented using one or more knowledge representation techniques. These techniques use different and interrelated *knowledge elements*. The knowledge elements range from *primitives* (including different forms of O-A-V triplets, frames, rules, logical expressions, and procedures), to *complex elements*. Complex knowledge elements are represented using either simple aggregations of knowledge primitives, or conceptually different techniques based on knowledge primitives and their combinations. In designing the entire knowledge base, all knowledge elements can be classified according to their *type* (e.g., rules, frames, or procedures) and grouped into several homogenous collections. Each such collection contains knowledge elements of the same type.
The rest of the paper is organized as follows. The next section surveys some knowledge modeling techniques that have received most attention in recent years by developers of intelligent systems, AI practitioners and researchers. Then another section discusses several practical systems, applications, and ongoing projects that use and implement those techniques. Finally, the paper briefly covers some of the most recent results in the fields of intelligent manufacturing systems, intelligent tutoring systems, and ontologies.

2. Concepts, Theory, Approaches, and Techniques

Along with the general major trends in computer engineering mentioned above, research and modeling efforts in some specific fields have helped to lay the ground for development of advanced practical intelligent systems. These fields include intelligent agents, ontologies and knowledge sharing,

knowledge processing, intelligent databases, knowledge discovery and data mining, distributed AI, knowledge management, and user modeling.

2.1. Intelligent Agents

Intelligent agent is a program that maps percepts to actions. It acquires information from its environment ("perceives" the environment) and decides about its actions and performs them.
While there is no real consensus about the definition of intelligent agents, the above one adapted from [53] is intuitively clear and essentially describes the general concept of *generic agent*, Figure 2. All more specific intelligent agents can be derived from that concept.

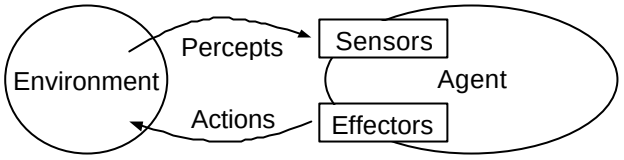

Figure 2. Generic agent (after [53])

There are different other names for intelligent agents, such as software agents, autonomous agents, adaptive interfaces, personal agents, network agents, softbots, knowbots, taskbots, and so on. Although there are minor differences among all these concepts, all of them are used to denote (one way or another) intelligent agents. For the purpose of this survey, we will adopt the term "intelligent agent" and its definition as an autonomous software entity that perceives its environment through sensors, acts upon it through its effectors, has some knowledge that enables it to perform a task, and is capable of navigation and communication with other agents.
Intelligent agents help their users locate, acquire, filter, evaluate, and integrate information from heterogeneous sources, i.e. coordinate information acquisition. Unlike most other kinds of intelligent systems, intelligent agents help *all categories* of end users. They help users in different ways, e.g. by hiding the complexity of difficult tasks, performing some tasks on behalf of their users, teaching end users, monitor events and procedures of interests to their users, helping the users collaborate and cooperate, and the like [26], [42]. Agents have the ability to identify, search, and collect resources on the Internet, to optimize the use of resources, and to perform independently and rapidly under changing conditions. However, the user doesn't necessarily "listen" to what the agent "says".
Intelligent agents are modeled after human agents. Typically, they interact with their users as cooperative assistants, rather than just letting the users manipulate them as traditional programs. They act autonomously or semiautonomously as communicational interfaces between human agents and other programs, as well as between other computational agents and other programs [44]. Along with the capabilities of autonomous operation and communication, their other useful properties include, initiative, timeliness of response, flexibility, adaptability, and often a learning capability. It should be understood, though, that the concept of intelligent agent is not an absolute characterization that divides the world to agents and non-agents.
From the architectural point of view, most agents fall into one of the following categories [27], [45], [62], [63]:

- *Logic-based agents*. Such agents make decisions about their actions through logical deduction, as in theorem proving. The internal state of a logic-based agent is assumed to be a database of formulae of classical first-order predicate logic. The database is the information that the agent has about its environment, and the agent's decision-making process is modeled through a set of deduction rules. The agent takes each of its possible actions and attempts to prove the formulae from its database using its deduction rules. If the agent succeeds in proving a formula, then a corresponding action is returned as the action to be performed.
- *Reactive agents*. Their decision-making is implemented in some form of direct mapping from situation to action, often through a set of task accomplishing behaviors. Each behavior may be thought of as an individual action function that maps a perceptual input to an action to perform. Many behaviors can fire simultaneously, and there is a mechanism to choose between the different actions selected by these multiple behaviors. Such agents simply react to their environment, without reasoning about it.
- *Belief-desire-intention (BDI) agents*. These agents internally represent their beliefs, desires, and intentions, and make their decisions based on these representations. BDI architectures apply practical reasoning, i.e. the process of continuously deciding which action the agent is to perform next in order to get closer to its goals. The process is represented in Figure 3, where a

belief revision function (*brf*) takes a sensory input and the agent's current beliefs, and on the basis of these, determines a new set of beliefs. Then an *option generation function* (*Generate options*) determines the options available to the agent (its desires), on the basis of its current beliefs about its environment and its current intentions. The *filter function* (*Filter*) determines the agent's intentions on the basis of its current beliefs, desires, and intentions, and an *action selection function* (*Action*) determines an action to perform on the basis of current intentions.

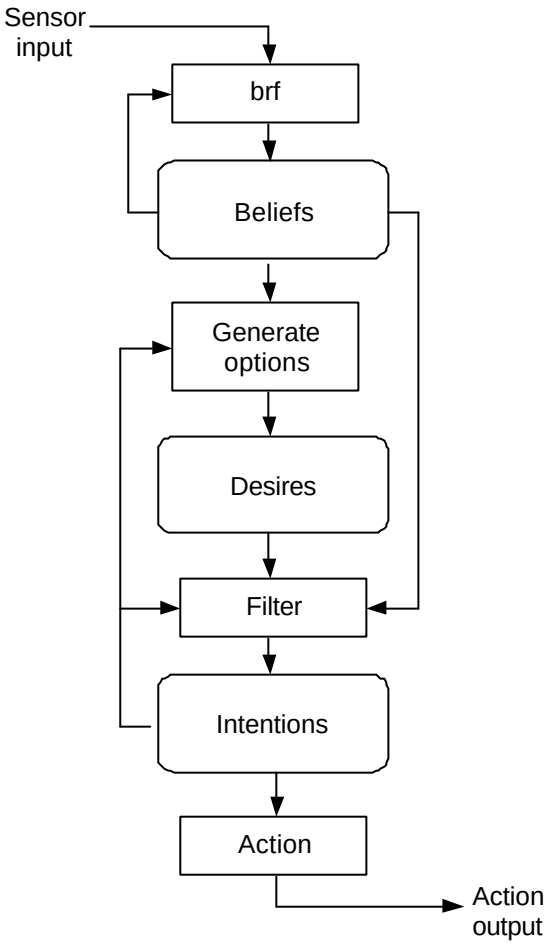

Figure 3. BDI agent architecture (after [63])

?? *Layered architectures*. Agents with layered architectures make their decisions via various software layers. Layers correspond to different levels of abstraction. Each layer supports more-or-less explicit reasoning about the environment. In *horizontally layered architectures*, Figure 4a, each layer is directly connected to the sensory input and action output, as if each layer itself was an agent. In *vertically layered architectures*, sensory input is performed by at most one layer, and so is action output. One typical version of vertically layered architectures is shown in Figure 4b. Here both the agent's sensors and effectors (not shown in Figure 4b) lay in the same layer, and the *API layer* is used to program them. Effectively, the API layer links the agent to the physical realization of its resources and skills. The *definition layer* contains the agent's reasoning mechanism, its learning mechanism (if it exists), as well the descriptions of the agent's goals, knowledge (facts), and the resources it uses in performing its task. The *organization layer* specifies the agent's behavior within a group of agents, i.e. what group the agent belongs to, what is the agent's role in the group, what other agents is the agent "aware of", and so on. The *coordination layer* describes the social abilities of the agent, i.e. what coordination/negotiation techniques it knows, such as coordination techniques for collaboration with other agents, techniques for exchanging knowledge and expertise with other agents, and techniques for increasing the group's efficiency in collaborative work. The *communication layer* is in charge of direct communication with other agents (exchanging messages). It handles low-level details involved in inter-agent communication.

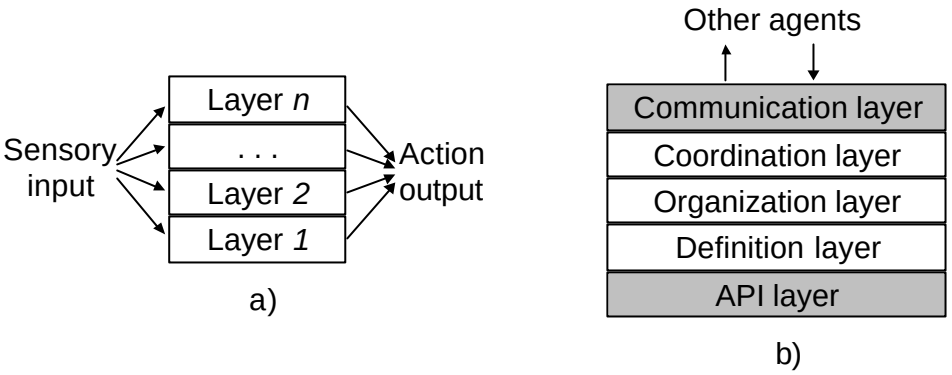

Figure 4. Layered architecture of intelligent agents a) horizontally layered architectures (after [62]) b) vertically layered architectures (after [45])

The communication layer in Figure 4a brings about the issue of agent communication. Collectively, agents may be physically distributed, but individually, they are logically distinct and unlikely to operate in isolation [39]. Agents typically communicate by exchanging messages represented in a standard format and using a standard *agent communication language* (*ACL*). An ACL must support a common syntax for agent communication, common semantics (i.e. domain ontologies as backbone of knowledge being communicated - see the next section), and common pragmatics (what agent talks, to what other agents, how to find a right agent to talk to (the identification problem), and how to initiate and maintain communication) [21]. A number of ACLs have been proposed so far. Two most frequently used are *KQML* (*Knowledge Query and Manipulation Language*) [21], [31], and *FIPA ACL* [22], [31].

KQML is a language and a set of protocols for agent communication that supports different architectures and types of agents. It also supports communication between an agent and other clients and servers. KQML uses standard protocols for information exchange (such as TCP/IP, email, HTTP, CORBA), and different modes of communication (synchronous, asynchronous, and broadcast). Figure 5 illustrates the layered organization of KQML messages, and Figure 6 shows an example KQML message. Message content can be described in any language and is wrapped-up in description of its attributes, such as the message type, the content language, and the underlying domain ontology. The outer wrapper is the "communication shell", specifying the message sender, the receiver, and the communication mode.

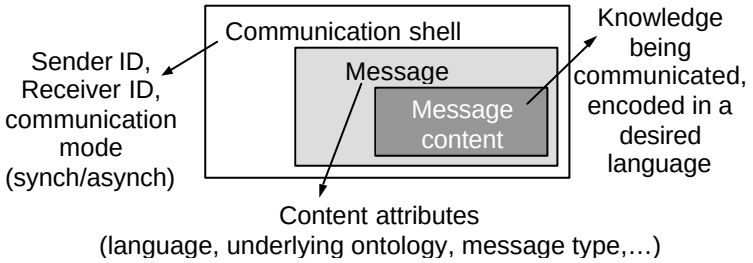

Figure 5. Layered organization of KQML messages

```
(MSG
        :TYPE query
        :QUALIFIERS (:number-answers 1)
        :CONTENT-LANGUAGE KIF
        :CONTENT-ONTOLOGY (blocksWorld)
        :CONTENT-TOPIC (physical-properties)
        :CONTENT (color snow ?C))
```

a)

```
(PACKAGE :FROM ap001
        :TO ap002
        :ID DVL-f001-111791.10122291
        :COMM sync
        :CONTENT
          (MSG
                :TYPE query
                :QUALIFIERS (:NUMBER-ANSWERS 1)
                :CONTENT-LANGUAGE KIF
                :CONTENT (color snow _C)))
```

b)

Figure 6. An example of KQML message (after [21]) a) the message content and attributes b) the message in its communication package

KQML also supports using *facilitators* and *mediators*, special-purpose agents that provide mediation and translation services for agent communication (e.g., forward messages to other agents, help "matchmaking" between information providers and servers, and maintain a registry of service names). Unlike direct, point-to-point communication between agents, facilitators and mediators serve as a kind of "event handlers" in a community of agents. They can register other agents' subscription of interest in certain information/knowledge and notify them when it becomes available. They can also collect other agents' "advertisements" of the information/knowledge they can provide, thus serving as information/knowledge brokers between the other agents. Moreover, facilitators and mediators can help other agents establish direct communication by providing negotiating services such as matchmaking (the identification problem).

FIPA's agent communication language is superficially similar to KQML. Like KQML, it is based on messages as actions, or *communicative acts*. The FIPA ACL specification consists of a set of message types and the description of their effects, as well as a set of high-level interaction protocols, including requesting an action, contract net, and several kinds of auctions [22]. FIPA ACL's syntax is identical to KQML's except for different names for some reserved primitives [31]. Like KQML, it separates the message's outer language from the inner language. The outer language defines the intended meaning of the message, and the inner one specifies the content. The inner language - Semantic Language, or SL - relies on the idea of BDI agents. SL is the formal language used to define FIPA ACL's semantics and is based on a quantified, multimodal logic with modal operators for beliefs, desires, uncertain beliefs, and intentions (persistent goals). SL can represent propositions, objects, and actions.

Two or more agents acting together form a *multiagent system* [39]. Centralized, "big" agents need huge knowledge base, create processing bottlenecks, and can suffer from information overload. In a multiagent system, several smaller agents divide complex tasks and specialize for performing parts of complex tasks. Multiagent systems achieve higher adaptivity of individual agents and the system as a whole, and higher flexibility when there are many different users.

Typical roles of agents in a layered, distributed, multiagent system, include *interface agents* (agents that receive tasks from the users and return results), *task agents* (agents that perform tasks received from interface agents), and *information agents* (agents that are tightly coupled with data sources). Some of the information agents facilitate matching of other agents. In such a system, agents run on different machines, information sources are also distributed, and agents cooperate in performing their tasks, thus creating a self-organizing system of agents that represents a shared resource for its users. Performance degradation of a multiagent system is graceful if an agent fails, because other agents in the system can often carry on the task of the problematic agent.

Mobile agents are units of executing computation that can migrate between machines [60]. The concept of mobile agents is based on remote programming (RP) for distributed systems, as opposed to remote procedure call (RPC) [2], [8], [61]. In the RPC case, Figure 7a, a client program issues a procedure/function call through a network to a remote server. In the RP case, the client program sends an agent to the remote server, where the agent issues local calls in order to obtain the desired service. After it completes its task on the remote server, it can get back to the client machine and deliver the results to the client program.

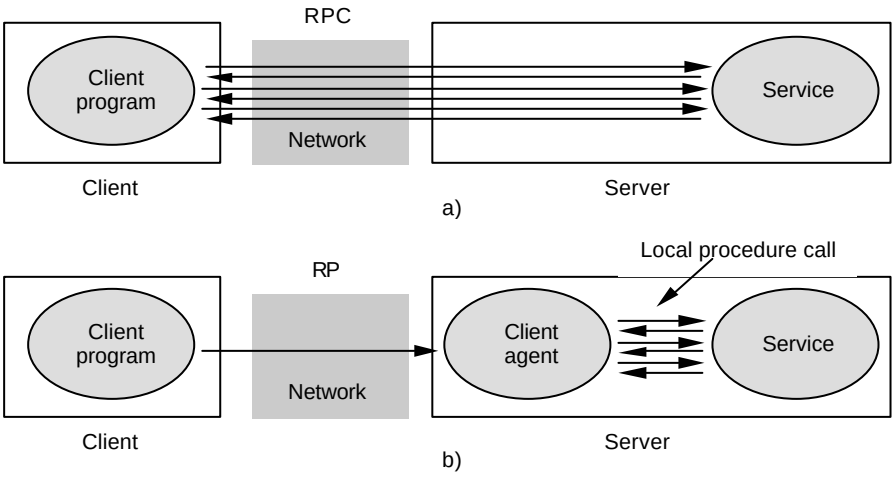

Figure 7. a) Remote procedure call b) Remote programming

Most mobile agents today are based on Java applets' mobility and Java RMI protocol for agent transport. There is a number shareware tools for mobile agents on the Internet, such as IBM Aglets.

Such tools make possible to create, deploy, deactivate, and delete mobile agents easily, often no direct coding. However, for both technical and security reasons, deploying mobile agents in practice usually requires an *agents server* to be installed on every machine that can be visited by a mobile agent created by using a specific tool. Agent server is the "gathering place" for mobile agents on a given machine. The agents work within that server, and migrate from one server to another carrying their states along. Mobile agents can move along a prespecified "itinerary", or can decide themselves dynamically where to migrate from a given server. The term *agents meetings* is used to denote collaborative work of several mobile agents within a given agent server that the agents previously negotiate about. In all such dislocated collaborative activities, a mobile agent can act as a "proxy" that brings the results of an agents meeting "home", i.e. to the home machine.

However, in spite of widespread use of mobile agents in e-commerce and in some tasks that are otherwise typically performed manually (such as automatic network configuration and control, to give but one example), the concept of mobile agents has been criticized for having some important constraints. The constraints include restrictions for meeting and collaboration mechanisms (everything must be done from within an agent server, and agent servers from different developers are often incompatible), critical data security (a mobile agent can steal data from a server, and a server can steal data from a mobile agent too!), and complex implementation of agent communication protocols and languages in case of mobile agents.

The concept of intelligent agents is a useful tool for system analysis and design. That fact has led to a growing adoption of intelligent agents in the software engineering community, and the terms like *agent-oriented programming* [29], [57], [60], *agent-oriented software engineering* [63], [64], and *agent-oriented middleware* [39] have been quickly coined.

One key idea of agent-oriented programming is that of directly programming agents in terms of "mentalistic notions" (such as belief, desire, and intention) [57], [63]. For example, the agents intentions, i.e. how the agent acts, can be expressed as a commitment rule set [57]. Each commitment rule contains a message condition, a mental condition, and an action. Matching the mental condition of a commitment rule against the beliefs of the agent and the message condition against the messages the agent has received, it is possible to decide whether the rule is satisfied. If it is, the agent becomes committed to the action. Actions may be private, corresponding to an internally executed subroutine, or communicative, i.e., sending messages. Message types can be defined starting from speech act theory. Some messages can modify the agent's commitments, and other can change its beliefs.

Other important ideas of agent-oriented programming and programming languages include providing support for issues like inter-agent communication, migration, and clean factorization of a system into its high-level application component and the infrastructure implementation [60]. In other words, agent-oriented programming at the low level requires entities like agents, sites, and communication channels, as well as primitives for agent creation, agent migration between sites, and location-dependent asynchronous message communication between agents. At the high level, it is necessary to represent location-independent communication between agents and provide infrastructure for a user-defined translation into the low-level primitives.

Agent-oriented software engineering draws its justification from the fact that intelligent agents represent a powerful abstraction, perhaps as powerful as procedural abstraction, abstract data types, and object-oriented programming [64]. In object-oriented software engineering, systems are modeled as a collection of interacting but passive objects. In agent-oriented software engineering, systems are understood and modeled as a collection of interacting autonomous agents. Thus, in a way, agent-oriented approach complements the object-oriented one. Although in agent-oriented software engineering agents are typically implemented using object-oriented techniques, there are usually fewer agents in the system than objects.

Of course, not every system can and not every system should be naturally modeled using agents. Agent-oriented software engineering is appropriate when building systems in which the agent metaphor is natural for delivering system functionality, data, control, expertise, or resources are distributed, or a number of legacy systems must be included and must cooperate with other parts of a larger system.

In order to gain wider acceptance for agent-oriented software engineering, appropriate extensions to several UML (Unified Modeling Language) diagrams have been proposed recently to express agent interaction protocols and other issues useful for modeling agents and agent-based systems [49]. An agent interaction protocol describes a communication pattern as an allowed sequence of messages between agents and the constraints on the content of those messages. Such communication patterns can be modeled as specific sequence diagrams, activity diagrams, and state charts in UML, possibly adorned with new notational elements (or appropriate variations of existing ones) to denote communication acts between the agents. The key idea here is to treat a whole agent interaction protocol as an entity in UML (a package or a template). Any such entity should be generic, in order to support

easy adaptations and specializations. Figure 8 illustrates how some proposed minor extensions to UML can be used to express agent communication. Here the *agent-name/role:class* syntax is an extension of of already part of *object-name/role:class* syntax that is already a part of UML the arrowed lines are labeled with an agent *communication act* (CA), instead of an OO-style *message*. Other extensions have been proposed as well to support concurrent threads of interaction, interaction among agents playing multiple roles, packages involving agents, and deployment diagrams for agents. Some of the extensions include specific UML stereotypes for agent-oriented design, such as <<clone>>, <<mitosis>>, <<reproduction>>, and the like.

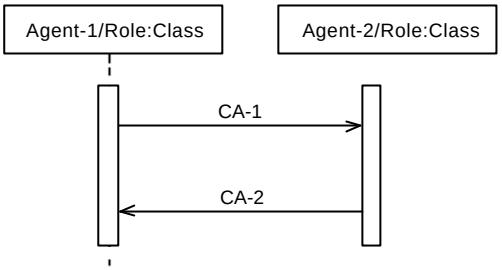

Figure 8. Basic format for agent communication (after [49])

Some reusability-related suggestions in agent-oriented software engineering include the following:

- ?? agents' design and implementation should be kept simple - this allows for easier customization to different users, easier reuse and maintenance;
- ?? many agents can be developed starting from copying code fragments from a similar agent's program, and then customizing them to different users.

If middleware is defined as any entity that is interposed between a client and a server, a peer and another peer, or an application and a network, then agent-oriented middleware should provide the following services to agent-based applications and systems [39]:

- ?? dynamic binding between agents and hardware they run on, including a handle for any entity/entities "owning" the agents;
- ?? location services, such as finding an appropriate agent to communicate with either directly or indirectly, maping task requests to service instances, and facilitating agent interaction in either client-server or peer-peer configurations; currently, facilitators and mediators are used in many agent applications as special-purpose agents that provide location services;
- ?? application services, including dynamic self-configuration and communication between agents and other applications;
- ?? registration and life-cycle management services; currently, these services and application services provide agent servers (at least to an extent).

Is middleware distinct from other agents or is it an agent itself? This open problem is resolved in practice using a design trade-off. In "fat middleware - thin agents" configurations, the abstraction of agents' common functionality is maximized and "delegated" to a separate middleware component. In "thin middleware - fat agents" configurations, each agent's autonomy is maximized and the common functionality delegated to the middleware is minimized. The tendency is toward more sophisticated agents and increased intelligence of an overall system. This moves the distribution of knowledge and intelligence toward the agents and away from the middleware, and essentially tums any middleware functionality into an agent.

2.2. Ontologies and Knowledge Sharing

In building knowledge-based systems, developers usually construct new knowledge bases from scratch. It is a difficult and time-consuming process. Moreover, it is usually hard to share knowledge encoded in such knowledge bases among different knowledge-based systems. There are several reasons for that. First, there is a large diversity and heterogeneity of knowledge representation formalisms. Even within a single family of knowledge representation formalisms, it can be difficult to share knowledge across systems [46]. Also, in order to provide knowledge sharing and reuse across multiple knowledge bases, we need standard protocols that provide interoperability between different knowledge-based systems and other, conventional software, such as databases. Finally, even if the other problems are eliminated, there is still an important barrier to knowledge sharing at a higher, *knowledge level* [47]. That is, there is often a higher-level modeling, taxonomical, and terminological mismatch of different systems, even if they belong to the same application domain.

Research in the growing field of *ontological engineering* [11], [17], [43], offers a firm basis for solving such problems. The main idea is to establish standard models, taxonomies, vocabularies and domain terminologies, and use them to develop appropriate knowledge and reasoning modules. Such modules would then act as reusable components that can be used for assembling knowledge-based systems (instead of building them from scratch). The new systems would interoperate with existing ones, sharing their declarative knowledge, reasoning services, and problem-solving techniques [11].
Ontologies, or explicit representations of domain concepts, provide the basic structure or armature around which knowledge bases can be built [59]. Each ontology is an explicit specification of some topic, or a formal and declarative representation of some subject area. It specifies 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. The ontology provides vocabulary (or names) for referring to the terms in that subject area, and the logical statements that describe what the terms are, how they are related to each other, and how they can or cannot be related to each other. Ontologies also provide rules for combining terms and relations to define extensions to the vocabulary, as well as the problem semantics independent of reader and context.
The purpose of ontologies is to enable knowledge sharing and reuse among knowledge based-systems and agents. Ontologies describe the concepts and relationships that can exist for an agent or a community of agents. Each such a description is like a formal specification of a program. In fact, a common ontology defines the vocabulary with which queries and assertions are exchanged among agents. Ontologies state axioms that constrain the possible interpretations for the defined terms.
If we think of ontologies in object-oriented way, then one possible interpretation of ontologies is that they provide taxonomic hierarchies of classes and the subsumption relation. For example, in the hierarchy of the Lesson class, we may have the Topic class, the Objective class, and the Pedagogical-point class. But on the other hand, we can develop Lesson, Topic, Objective, and Pedagogical-point ontologies as well. In that case, the Lesson ontology would subsume the other three ontologies.
Ontologies make possible to define an infrastructure for integrating intelligent systems at the knowledge level [53]. The knowledge level is independent of particular implementations. It defines adequate representation primitives for expressing knowledge in a convenient way (usually those used by a knowledge engineer) [16]. The representation primitives can be regarded as an ontology (as concepts and relations of a particular domain), suitable for defining a knowledge representation language. Once the ontology is formalized, it defines the terms of the representation language in a machine-readable form.
Ontologies are especially useful in the following broad areas:

- collaboration - ontologies provide knowledge sharing among the members of interdisciplinary teams and agent-to-agent communication;
- interoperation - ontologies facilitate information integration, especially in distributed applications;
- education - ontologies can be used as a publication medium and a source of reference;
- modeling - ontologies represent reusable building blocks in modeling systems at the knowledge level.

Ontologies are also expected to play a major role in *The Semantic Web* [16]. The Semantic Web is the next-generation Web that will enable automatic knowledge processing over the Internet, using intelligent services such as search agents, information brokers, and information filters. Doing this will require to define standards not only for the syntactic form of documents, but also for their semantic content, in order to facilitate semantic interoperability. *XML* (*eXtended Markup Language*) and *RDF* (*Resource Description Framework*) [3], [13], [33], [48] are the current World Wide Web Consortium standards for establishing semantic interoperability on the Web. However, although XML has already been successfully used to represent ontologies, it is more syntactically oriented, addresses only document structure (just describes grammars), and provides no way to recognize a semantic unit from a particular domain. On the other hand, RDF provides a data model that can be extended to address sophisticated ontology representation techniques, hence it better facilitates interoperation. In fact, RDF has been designed to standardize the definition and use of metadata - descriptions of Web-based resources - but is equally well suited to representing data. Its basic building block is object-attribute-value triplet, which is convenient for representing concepts in ontologies. Moreover, a domain model - defining objects and relationships - can be represented naturally in RDF. Defining an ontology in RDF means defining an RDF schema, which specifies all the concepts and relationships of the particular language [16]. RDF schema mechanism can be also used to define elements of an ontology representation and inference language.

2.3. Knowledge Processing

We can define a *knowledge processor* as an abstract mechanism which, starting from a set of given facts and a set of given knowledge elements produces some changes in the set of facts. Concrete examples of knowledge processors include (but are not limited to) blackboard control mechanisms, heuristic classifiers, rule-based inference engines, pattern-matchers, and at the lowest level even a single neuron of a neural network. As an illustrative exa mple, consider the blackboard architecture in Figure 9. It models solving a complex problem though cooperation and coordination of a number of specialists, called knowledge sources (KS). Each knowledge source is specialized in performing a certain task that can contribute to the overall solution, and all the knowledge sources share partial results of their processing through a shared memory, called the blackboard. Knowledge sources are independent in their work to a large extent, since their knowledge resides in their local knowledge bases and many of the facts they use is local to them. Hence most processing/reasoning of a knowledge source is done within the knowledge source itself. A global blackboard control mechanism coordinates the work of individual knowledge sources and synchronizes their access to the blackboard.

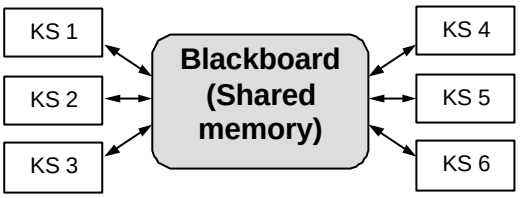

Figure 9. The blackboard architecture

Two important lines of developments in AI regarding knowledge processing have emerged during the last decade. The first one is development of different kinds of knowledge processors starting from well-established software design practices. An example is using *software patterns* to specify common global architecture of different knowledge-processing mechanisms [17]. In software engineering, patterns are attempts to describe successful solutions to common software problems [54]. Software patterns reflect common conceptual structures of these solutions, and can be applied over and over again when analyzing, designing, and producing applications in a particular context.
Figure 10 shows the structure of the *Knowledge processor pattern* [18]. Its participants have the following meanings. *Knowledge processor* defines an interface for using the knowledge from the *Knowledge* object, as well as for examining and updating facts in the *Facts* object. *Knowledge* and *Facts* objects are generally aggregates of different collections of knowledge elements. Parameterizing collections of knowledge elements, we can actually put collections of rules, frames, etc. in the *Knowledge* object, thus making it represent a knowledge base. By analogy, we can also make the *Facts* object represent a working memory, containing collections of working memory elements, rule and frame instantiations, etc. *Knowledge processor* contains also a pointer to an instantiation of the abstract *Interface* class. Developers can subclass *Interface* in order to implement an application-specific interface to a particular knowledge processor. *Concrete Knowledge Processor* is either a knowledge processor of a specific well-known type (e.g., a forward-chaining inference engine, a fuzzy-set-based reasoner, a knowledge-based planner), or can be defined by the application designer.

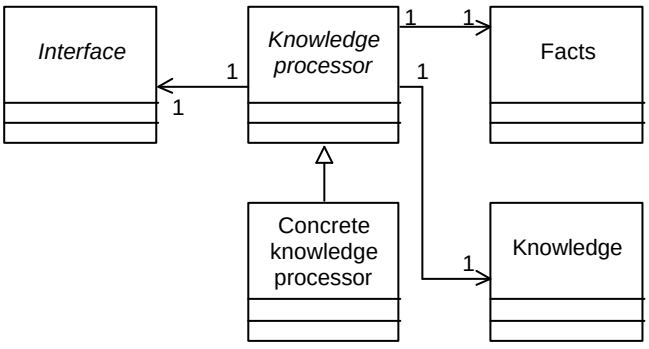

Figure 10. The Knowledge processor pattern

The second important line of developments in AI regarding knowledge processing is the line of integrating traditional programming languages with knowledge-processing capabilities. Examples include the Tanguy architecture [14] and the Rete++ environment [24]. The Tanguy architecture

extends the C++ language to cope with permanent object storage, production rules (data-driven programming), and uniform set-oriented interfaces. The Rete++ environment embeds pattern matching based on the famous Rete algorithm into C++.

2.4. Intelligent Databases

One way to make data access and manipulation in large, complex databases simple and more efficient is to integrate database management systems with knowledge processing capabilities. In that way, database designers create *intelligent databases*. They are featured by query optimization, intelligent search, knowledge-based navigation through huge amounts of raw data, automatic translation of higher-level (natural language) queries into sequences of SQL queries, and the possibility of making automatic discoveries [51].
Intelligent databases have evolved through merging of several technologies, including traditional databases, object-oriented systems, hypermedia, expert systems and automatic knowledge discovery. The resulting top-level, three-tier architecture of an intelligent database, Figure 11, has three levels: high-level tools, high-level user-interface, and intelligent database engine. *High-level tools* perform automated knowledge discovery from data, intelligent search, and data quality and integrity control [51]. The users directly interact with the *high-level user interface*. It creates the model of the task and database environment. The *intelligent database engine* is the system's base level. It incorporates a model for a deductive object-oriented representation of multimedia information that can be expressed and operated in several ways.

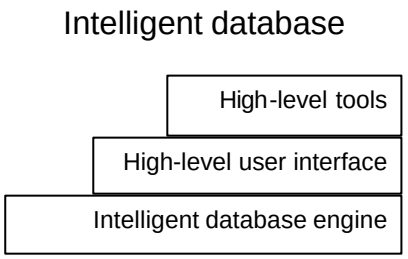

Figure 11. Three-tier architecture of intelligent databases

Typical concrete ways of merging database technology with intelligent systems are coupling and integration of DBMSs and intelligent systems. *Coupling* is a weaker merger. It does not guarantee consistency of rules and data (a database update may not go through the intelligent system). It also raises difficulties when trying to bring the database into conformity with new rules in the knowledge base. In spite of that, there are many successful commercial applications that use coupling of intelligent systems and databases. In such applications the intelligent systems play the role of intelligent front-ends to databases. On the other hand, *integration* of DBMSs and intelligent systems guarantees consistency of rules and data, because a single DBMS administers both kinds of objects. Moreover, integration usually brings better performance then mere coupling. There are very well known examples of integration of rules and data in commercial DBMSs, e.g. INGRES and Sybase.

2.5. Knowledge Discovery and Data Mining

Knowledge Discovery in Databases (KDD) is the process of automatic discovery of previously unknown patterns, rules, and other regular contents implicitly present in large volumes of data. *Data Mining (DM)* denotes discovery of patterns in a data set previously prepared in a specific way. DM is often used as a synonymous for KDD. However, strictly speaking DM is just a central phase of the entire process of KDD.
Knowledge discovery is a *process*, and not a one-time response of the KDD system to a user's action. As any other process, it has its environment, its phases, and runs under certain assumptions and constraints.
Figure 12 illustrates the environment of the KDD process [40]. The necessary assumptions are that there exists a database with its data dictionary, and that the user wants to discover some patterns in it. There must also exist an application through which the user can select (from the database) and prepare a data set for KDD, adjust DM parameters, start and run the KDD process, and access and manipulate discovered patterns. KDD/DM systems usually let the user choose among several *KDD methods*. Each method enables data set preparation and search in order to discover/generate patterns, as well as pattern

evaluation in terms of certainty and interestingness. KDD methods often make possible to use domain knowledge to guide and control the process and to help evaluate the patterns. In such cases domain knowledge must be represented using an appropriate knowledge representation technique (such as rules, frames, decision trees, and the like). Discovered knowledge may be used directly for database query from the application, or it may be included into another knowledge-based program (e.g., an expert system in that domain), or the user may just save it in a desired form. Discovered patterns mostly represent some previously unknown facts from the domain knowledge. Hence they can be combined with previously existing and represented domain knowledge in order to better support subsequent runs of the KDD process.

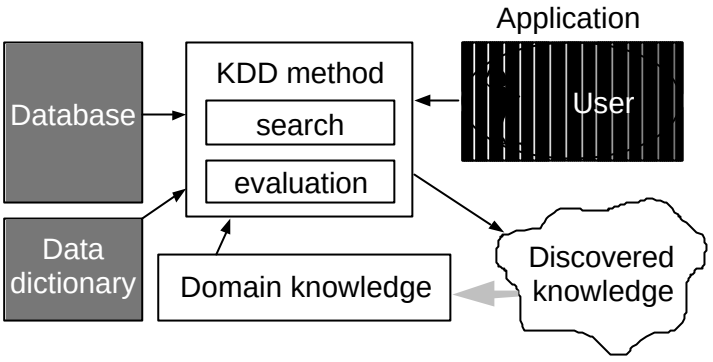

Figure 12. Environment of the KDD process (after [40])

Figure 13 shows typical activities, phases, and data in the KDD process [20]. KDD is never done over the entire *database*. Instead of that, a representative *target data set* is generated from a large database by an appropriate *selection* procedure. In the next phase, *preprocessing* of target data is necessary in order to eliminate noise (handle missing, erroneous, inexact, imprecise, conflicting, and exceptional data, resolve ambiguities in the target data set) and possibly further prepare target data in terms of generating specific data sequences. The result is the set of *preprocessed data*.

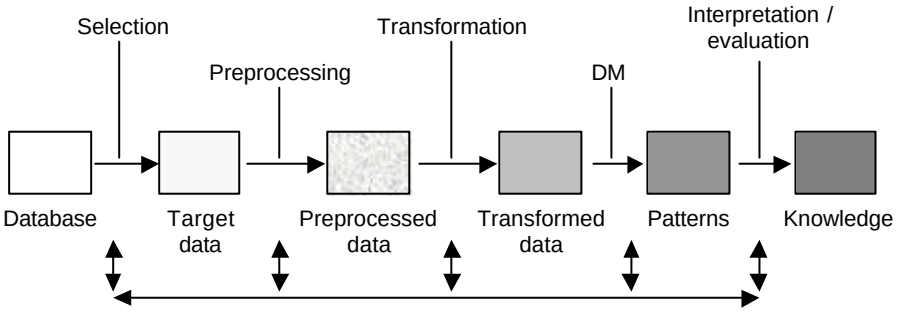

Figure 13. Phases in the KDD process (after [20])

The next phase is *transformation* of the preprocessed data into a suitable form for performing the desired *DM task*. DM tasks are specific kinds of activities that are carried out over the set of *transformed data* in search of patterns, guided by the kind of knowledge that should be discovered. Some examples of DM tasks are classification, cluster identification, mining association rules, mining path-traversal patterns, change and deviation detection, and sequence analysis. The output of the *DM* phase is, in general, a *set of patterns*. However, not all of the patterns are useful. The goal of *interpreting and evaluating* all the patterns discovered is to keep only those patterns that are interesting and useful to the user and discard the rest. Those patterns that remain represent the discovered *knowledge*.

In practice, KDD process never runs smoothly. On the contrary, it is a time-consuming, incremental, and iterative process by its very nature, hence many repetition and feedback loops in Figure 13. Individual phases can be repeated alone, and the entire process is usually repeated for different data sets.

Discovered patterns are usually represented using a certain well-known knowledge representation technique, including inference rules, decision trees, tables, diagrams, images, analytical expressions, and so on. Inference rules (If-Then rules) are the most frequently used technique. Decision trees are a suitable alternative, since with many machine learning algorithms the concepts the program learns are represented in the form of decision trees. Transformations between decision trees and inference rules

are easy and straightforward. Rules are often dependent on each other, so the discovered knowledge often has the form of causal chains or networks of rules.
KDD systems usually apply some probabilistic technique to represent uncertainty in discovered patterns. Some form of certainty factors often goes well with inference rules. Probability distribution is easy to compute statistically, since databases that KDD systems start from are sufficiently large. Probability distribution is especially suitable for modeling noise in data. Fuzzy sets and fuzzy logic are also used sometimes. However, it is important to note that an important factor in modeling uncertainty is the effort to actually *eliminate* sources of uncertainty (such as noisy and missing data) in early phases of the process.
What technique exactly should be used to represent discovered knowledge depends on the goals of the discovery process. If discovered knowledge is for the users only, then natural language representation of rules or some graphically rich form is most suitable. Alternatively, discovered knowledge may be used in the knowledge base of another intelligent application, such as an expert system in the same domain. In that case, discovered knowledge should be translated into the form used by that other application. Finally, discovered knowledge may be used along with the previously used domain knowledge to guide the next cycle of the KDD process. That requires representing discovered patterns in the same way the other domain knowledge is represented in the KDD system.

2.6. Distributed Intelligent Systems

Distributed AI systems are concerned with the interactions of groups of intelligent agents that cooperate when solving complex problems. Distributed problem solving, as a subfield of AI, deals with strategies by which the decomposition and coordination of computation in a distributed system are matched to the structural demands of the task domain. Distributed intelligent systems model a number of information processing phenomena that occur in the natural world. Such phenomena are a source of a number of useful metaphors for distributed processing and distributed problem solving.
Recent developments in distributed systems in general, and particularly the Internet, have further contributed to the importance of distributed intelligent systems. The increase of information technology capabilities due to the development of the Internet and the World Wide Web has made possible to develop more powerful and often widely dispersed intelligent systems. Although these new systems often merely implement in a new way some well-established AI techniques, such as planning, search, and problem solving, they definitely have their own identity. Technologies such as intelligent agents, knowledge servers, virtual organizations, and knowledge management (to name but a few) are tightly coupled with the Internet, and have opened new fields for application and practical use of AI during the last decade.
Since intelligent agents and knowledge management are covered with more details in the other dedicated sections of this paper, this section covers knowledge servers and virtual organizations more extensively.
A *knowledge server* is an entity in a computer network (most often on the Internet) that provides high-level, high-quality, knowledge-based services such as expert advice, heuristic search and adaptive information retrieval, automatic data interpretation, portals for knowledge sharing and reuse, intelligent collaboration, and so on. For example, an expert system on the Internet that performs its tasks remotely can be thought of as a knowledge server. Users access the system from a distant location and get expert problem solving over the Internet. As another example, viewing intelligent agents as servers with appropriate knowledge is sometimes also convenient. Recall that all intelligent agents posses some knowledge and some problem-solving capabilities. Deploying agents to do some work on behalf of their users, communicating with other agents in a network environment, can be interpreted as knowledge-based assistance (service) to remote users.
Communicating with a knowledge server from a remote computer necessarily requires an appropriate software front-end. Nowadays such front-ends are most often Java-based interfaces. Figure 14 illustrates how a knowledge server and its user-interface front-end application operate in an essentially client-server manner. Note, however, that dividing a previously developed, integrated intelligent system into a client and a server in order to operate on a network (Internet) can be difficult. The most important reason is the need to minimize communication between the client and the server and ensuring acceptable response time. As Figure 14 shows, the solution is usually to let the client perform computationally expensive user-interface operations, and let the knowledge server both store the knowledge base and perform the problem-solving task locally. A mobile agent can be deployed to provide appropriate interaction between the client and the server. Ensuring multiplatform support in such applications is a must, so Java virtual machine usually comes as a rescue.

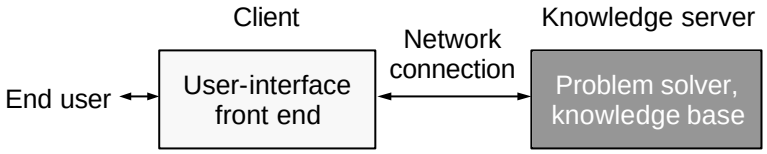

Figure 14. Client-server architecture of knowledge servers

If multiple knowledge servers are to cooperate in solving a complex problem, the architecture is more complex and calls for knowledge sharing between multiple servers. This is just another problem in the field of intelligent systems that stresses the need for their ontological foundation.
Practical problems of using knowledge servers are those of network congestion, unreliable network connection, and server availability. Installing mirror sites for knowledge servers can increase server robustness and availability.
In a *virtual organization*, complementary resources existing in a number of cooperating companies are left in place, on their servers, but are *integrated* to support business processes or a particular product effort [36]. The resources get *selectively assigned* to the virtual organization if their "owner" does not use them. Moreover, the resources in a virtual organization get *quickly* created, assembled and integrated within a business domain. Virtual organizations use the Internet selectively in order to create or assemble productive resources quickly, frequently and concurrently. Such productive resources include research, manufacturing, design, business, learning and training, etc. AI has been successfully used in a number of virtual organizations, including virtual laboratories, virtual office systems, concurrent engineering projects, virtual classrooms, and virtual environments for training.
Virtual organizations use the diffusion of computer networks and the globalization of specialized knowledge to become independent of geographical constraints [30]. They have restructured around communication networks, building on organizational principles that emphasize flexibility, knowledge accumulation and deployment, and distributed teamwork. Applying intelligent agents in such a setting enables collaboration between the users and resource agents in a virtual organization, provides knowledge exchange about the resources, and facilitates user interaction with these resources. In fact, in virtual organizations intelligent agents, knowledge-based systems, and AI-based planning become essential, since humans have limited ability to keep track of what is going on in the broad range of virtual organization activities, given the tight time constraints and limited resources required and used by virtual organizations [36]. Worse still, there is a frequent interruption of their work; for example, white-collar employees receive a communication (electronic, paper, or oral) every five minutes. Using agents and other AI technologies in virtual organizations mitigates the limitations and constraints of humans and makes possible to monitor and control substantial resources without the time constraints inherent in human organizations.

2.7. Knowledge Management

Knowledge management is the process of converting knowledge from the sources available to an organization and connecting people with that knowledge [35]. It involves the identification and analysis of available and required knowledge assets and knowledge asset related processes, and the subsequent planning and control of actions to develop both the assets and the processes so as to fulfill organizational objectives [1]. At the Artificial Intelligence Applications Institute of the University of Edinburgh, they define knowledge assets as the knowledge regarding markets, products, processes, technologies and organizations, that a business owns or needs to own and which enable its business processes to generate profits, add value, etc. Knowledge management involves not only these knowledge assets but also the processes that act upon the assets, such as developing knowledge, preserving knowledge, using knowledge, assessing knowledge, applying knowledge, updating knowledge, sharing knowledge, transforming knowledge, and knowledge transfer. Knowledge management facilitates creation, access, and reuse of knowledge, typically using advanced technology, such as World Wide Web, Lotus Notes, the Internet, and intranets. Knowledge-management systems contain numerous knowledge bases, with both numeric and qualitative data and knowledge (e.g., searchable Web pages). Important AI developments, such as intelligent agents, knowledge discovery in databases, and ontologies, are also important parts of knowledge-management systems.
Formal management of knowledge assets and the corresponding processes is necessary for several reasons. In the fast changing business world, knowledge needs to evolve and be assimilated also fast [1]. On the other hand, due to competitive pressures the number of people who hold this knowledge is decreasing. Knowledge takes time to experience and acquire, and employees have less and less time for this. When employees retire or leave the company, as well as when the company undergoes a change in

strategic direction, some knowledge gets lost. Within the company, all the important knowledge must be represented using a common vocabulary if the knowledge is to be properly understood. In order to be able to share and reuse knowledge among differing applications in the company and for various types of users, it is necessary to share the company's existing knowledge sources, and to identify, plan, capture, acquire, model, validate, verify and maintain future knowledge assets. In everyday practice, the company's employees need access to the right knowledge, at the right time, in the right location. Essentially, all this stresses the need to model business processes, the resources, the capabilities, roles and authority, the communication between agents in the company, and the control over the processes. Knowledge modeling and knowledge engineering practices help achieve these objectives [38]. Figure 15 illustrates what IT/AI technologies are the major knowledge management enablers.

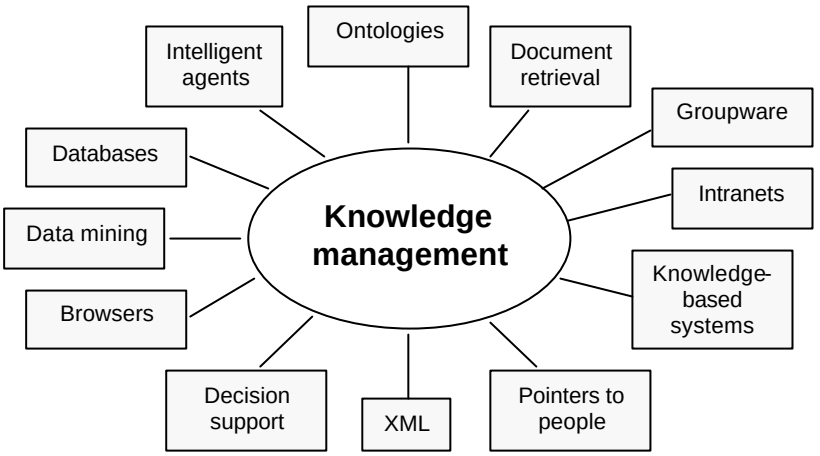

Figure 15. Key IT/AI eneblers for formal knowledge management

Numerous KM applications have been developed for enhancing customer-relationship management, research and discovery, and businesses processes, as well as group collaboration for design and other purposes [34]. Their typical architecture is layered, starting from *information and knowledge sources* at the bottom (word processing, electronic document management, databases, email, Web, people) and an appropriate *infrastructure* layer on top of the base one (email, file servers, Internet/intranet services). The adjacent upper layer is *knowledge repository* that provides content management. Further up from it is the *knowledge map*, defining corporate taxonomy, and the *knowledge management services* layer is on top of it. It contains two kinds of services, *discovery services* and *collaboration services*. They are just beneath the *knowledge portal* layer, that provides applications' interface to the knowledge management system. *Application layer* on top of all that specifies the company's applications that use the knowledge management system. These may include customer-relationship management, competitive intelligence, best-practice systems, product development and other applications. Obviously, knowledge management is not a single technology but instead is a collection of indexing, classifying, and information-retrieval technologies coupled with methodologies designed to achieve results desired by the user.

The conventional approach to knowledge management keeps data in a database. In this case, the database schema provides the context for the data. However, this approach suffers from poor scalability. A convenient alternative is to use XML (eXtensible Markup Language) instead, so that XML provides the context for turning a data set into information, and knowledge management provides the tools for managing this information [13].

Superficially, XML is a markup language like HTML. A deeper insight shows that XML is actually a family of languages that provide a more semantic management of information than HTML [3]. It has been designed and published by World Wide Web Consortium as a standard that provides means to describe data and unlike HTML, XML tags are not predefined in XML [48]. XML is self-describing. It uses a Document Type Definition (DTD) to formally describe the data. XML is a universal language for data on the Web that lets developers deliver content from a wide variety of applications to the desktop [52]. XML promises to standardize the way information is searched for, exchanged, adaptively presented, and personalized. Content authors are freed from defining the style of the XML document as the content has been separated out from the presentation of the document. Extensible Stylesheet Language (XSL) is useful in displaying the information in many ways e.g., in the form of a table, a bulleted list, a pie chart, or even in a different language.

In the context of knowledge management, XML is best thought of as a set of rules for defining data structures. Key elements in a document can be categorized according to meaning, rather than simply

how they are presented in a particular browser [52]. Instead of a search engine selecting a document by the metatags listed in its header, a search engine can scan through the entire document for the XML tags that identify individual pieces of text and image. Defining objects that tag or identify details about data is increasingly popular with companies trying to leverage both the structured data in relational databases and the unstructured data found on the Internet and in other sources. Knowledge management is trying to bring all this data together, and XML categorizes it [13]. Data modelers produce an XML DTD, which represents this domain data model. Programmers then provide the knowledge management tools with reusable XML parser code so that the community members can easily interact with the model. The knowledge management tools are being used to pass information to one another or to move it between knowledge repositories. The KM tools have data and context together by tagging the exported data with XML syntax or by interpreting the imported-tagged data according to the already created domain DTD. Knowledge repositories store the data either as marked-up flat files or in databases with schemas that are consistent with the domain DTD. The data is never separated from its context, and the context always accurately represents the domain data model.
A fundamental building block of a knowledge management infrastructure and architecture is *knowledge portal*. Generally, portals are large doors or gateways to reach many other places, indicating that the portal itself is not the final destination. A Web portal is a web site, usually with little content, providing links to many other sites that can either be accessed directly by clicking on a designated part of a browser screen, or can be found by following an organized sequence of related categories. A knowledge portal provides two distinct interfaces: a *knowledge producer interface*, supporting the knowledge mapping needs of the knowledge worker in their job of gathering, analyzing, adding value, and sharing information and knowledge among peers, and a *knowledge consumer interface* that facilitates the communication of the product of the knowledge workers and its dissemination through the enterprise to the right people, at the right time, to improve their decision-making. Knowledge portal provides all the facilities of an information catalog plus collaborative facilities, expertise management tools, and a knowledge catalog (access to the knowledge repository). The knowledge catalog is a metadata store that supports multiple ways of organizing and gathering content according to the different taxonomies used in the enterprise practice communities, including an enterprise wide taxonomy when defined.

2.8. User Modeling

A user model is an explicit representation of properties of a particular user, which allows the system to adapt diverse aspects of its performance to individual users. Adapting the definition from [58], user modeling can be thought of as the construction of a qualitative representation, called a *user model,* that accounts for *user behavior* in terms of a system's *background knowledge*. These three - the user model, the user behavior, and the background knowledge - are the essential elements of user modeling.
Techniques for user modeling have been developed and evaluated by researchers in a number of fields, including artificial intelligence, education, psychology, linguistics, and human-computer interaction. As a matter of fact, the field of user modeling has resulted in significant amounts of theoretical work, as well as practical experience, in developing applications that "care" about their users in traditional areas of human-computer interaction and tutoring systems [55]. It also has a significant impact on recent developments in areas like information filtering, e-commerce, adaptive presentation techniques, and interface agents [7].
In user modeling, user behavior is the user's observable response to a stimulus from the application that maintains the student model. The user behavior is typically an action (e.g., completing a form on the Web) or, more commonly, the result of that action (e.g., the completed form). For example, while a user is "browsing" through an adaptive hyperdocument all user actions are registered [6]. Based on these observations, the user modeling system maintains a model of the user's knowledge about each domain model concept. In the case of adaptive hyperdocument systems, for each domain model concept the user model keeps track of how much does the user know about this concept and whether the user has read something about this concept. Some user modeling systems can construct a user model from a single piece of behavior, while some other require multiple behaviors to accomplish their task.
The background knowledge comprises the correct facts, procedures, concepts, etc. of that domain, as well as the misconceptions held and other errors made by a population of users in the same domain [58]. Other background knowledge may include historical knowledge about a particular user (e.g., past qualitative models and quantitative measures of performance, user preferences and idiosyncracies, etc.), and stereotypical knowledge about user populations in the domain.

Being the output of the user modeling process, the user model contains partial, primarily qualitative representation of the user's knowledge (beliefs) about a particular topic, skill, procedure, domain, or process. The user model describes objects and processes in terms of spatial, temporal, or causal relations. In tutoring systems, the user modeling process detects mismatches between actual and desired behaviors and knowledge in terms of incorrect or inconsistent knowledge (i.e., misconceptions) and missing or incomplete knowledge. Intelligent analysis of the user's actions has to decide whether the actions are correct or not, find out what exactly is wrong or incomplete, and possibly identify which missing or incorrect knowledge may be responsible for the error (the last functionality is referred as knowledge diagnosis) [7]. Intelligent analyzers can provide the user with extensive error feedback and update the user model. A more convenient approach is to use interactive intelligent help support that provides intelligent advice to the user on each step of his/her interaction with the application. The level of help can vary from signaling about a wrong step, to giving a hint, to executing the next step for the user. In any such a case, the user modeling system watches the actions of the user, understands them, and uses this understanding to provide help and to update the user model.
The field of user modeling is very popular and constantly growing in recent years. There is a number of interesting research issues, such as:

- ?? construction of user models: knowledge, beliefs, and misconceptions, preferences, goals and plans, cognitive styles, user modeling agents and brokers;
- ?? exploitation of user models to achieve: adaptive information filtering and retrieval, tailored information presentation, transfer of task performance from user to system, selection of instructional actions, interface adaptation;
- ?? integrating active intelligent agents into user modeling systems, in order to let an active intelligent agent be a guide or assistant to each individual user and attempt to take into account the current progress of the user in the application's domain or skills, the specific goal/task of the user, the specific needs of the user to communicate with other users, and the perceived mental model of the user in the charting of a personalized use of the system;
- ?? applying user modeling agents in collaborative environments and work scenarios; that is, integrating the ideas of interactive learning environments, collaborative systems, distributed systems, and open systems;
- ?? cooperative user models, i.e. constructing the user models for a system/application in collaboration with each user;
- ?? using statistical techniques for predicting the user's ability and update the user model accordingly, based on the number of hints the user has required up to a certain point in interacting with the system;
- ?? inference techniques for user modeling, including neural networks, numerical uncertainty management, epistemic logic or other logic-based formalisms, stereotype or task hierarchies;
- ?? creating and maintaining appropriate user models for Web-based adaptive systems;
- ?? integration of individual user models in collaborative work environments in order to create multiple user models or group models;
- ?? applications of user modeling techniques in various areas, like adaptive learning and on-line help environments, e-commerce, interface agents, explanation of system actions, adaptive hypermedia and multimodal interaction, support of collaboration, of users with special needs;
- ?? practical issues of user modeling, like privacy, consistency, evaluation, standardization.

3. Projects, Systems, and Applications

There are a number of practical AI systems, applications and development projects that employ the knowledge modeling techniques surveyed in the first part of this paper. The sections of this second part present some concrete examples of such projects, systems, and applications.

3.1. I3 program

I3 stands for *Intelligent Integration of Information*. It is a long-term, DARPA-sponsored research and development program, concerned with the development of large-scale, distributed intelligent applications [37]. I3 uses several specific science areas, such as knowledge-base management, object-oriented software development, and meta-language descriptions. It builds upon earlier work established by DARPA on knowledge representation and communication standards.

The main idea of the I3 project is to set sandards for transforming dispersed collections of heterogeneous data sources into *virtual knowledge bases*. The sources of data include databases, knowledge bases, sensor-based subsystems and simulation systems, scattered over different servers on the Internet and intranets. Virtual knowledge bases integrate the semantic content of such disparate sources and provide integrated information to end-user applications at the right level of abstraction, Figure 16. A whole family of intelligent agents is in charge of the mediation process. The agents are hierarchically organized and coordinated as a group of different facilitators, mediators, and translators. As a result, the user can access the data sources at a high, user-oriented level, and get the desired abstracted information without worrying at all about the underlying translation and mediation processes. Moreover, the distributed nature of the family of agents eliminates the need for having a number of human and machine resources as intermediary information translators and processors.

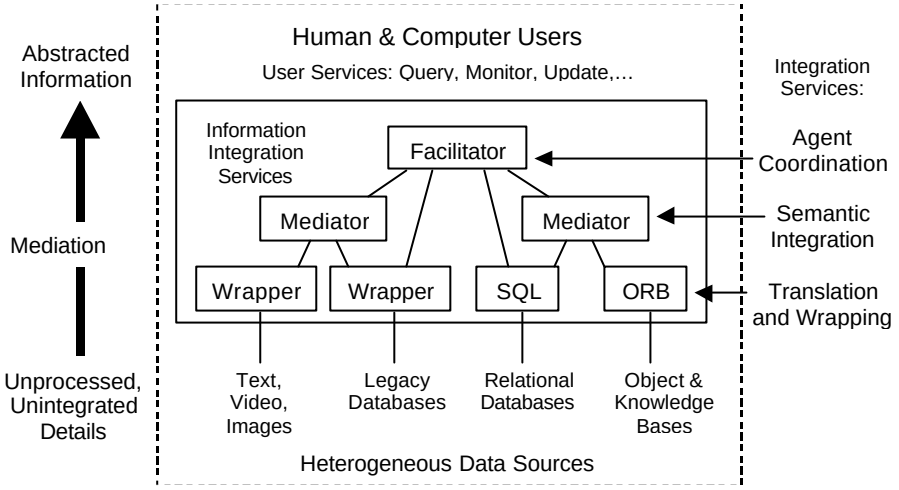

Figure 16. The I3 process

Turning Figure 16 "upside down" and looking inside the I3 services reveals the details of this multilayered architecture, Figure 17. I3 service layers isolate applications from dynamic changes in the real-world environment and the rigidity of legacy data. The services are intended to be used by both research and application communities to design and integrate I3 components. The I3 framework defines a set of essential I3 objects that are used in the protocol of these services (e.g., Information request object, Repository object, and Ontology object). Information Services form the base level of information processing. They are responsible for interaction with client components. Repository Services manage storage and representation of objects, while Mediation Services handle queries (translation, brokering, and decomposition), data access through protocols specific to data sources, and integration of objects from multiple information sources in possibly domain dependent ways. Examples of this kind of integration include unit conversions, data calculations, name correlation, and the like. I3 also defines Ontology Services, specific to knowledge modeling. These services include ontology viewer/editor, domain model viewer/editor, merging of concepts, inference and classification. A dedicated working group within the program is concerned with the development of an ontology meta-language (an "ontology ontology") for use in constructing sentences about ontology components.

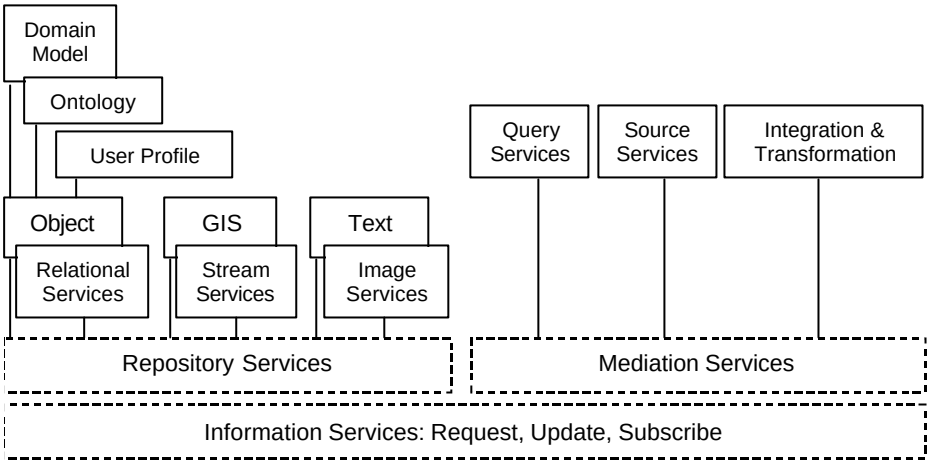

Figure 17. I3 service layers

3.2. Simulation based design - SBD system

A domain-independent, distributed intelligent system/tool for simulation-based design of engineering products, is developed by Lockheed Martin [15]. The system is called SBD (for Simulation-Based Design). It is an implementation of a domain-independent concurrent engineering framework focusing on fundamental engineering processes. These processes include product and process representation, collaboration and design process assembly and activation, visualization and interaction with product data, and integration of external applications. SBD system provides agent-based support for all of these processes. SBD is a concrete example of how the I3 framework, architecture and services are used. It also has many attributes of a virtual organization.

As a multi-agent, distributed, collaborative, virtual development environment with knowledge-based communications among its agents, SBD system is applicable throughout the product lifecycle. Due to its full-scale interoperability, it can be used in a distributed heterogeneous computing environment. In fact, SBD supports development of virtual prototypes of products and processes, as well as evaluation of these prototypes in synthetic environments that represent the entire lifecycle of the products. It is adaptable to many specific product domains.

The overall architecture of the SBD system is shown in Figure 18. It provides different kinds of services that can be matched to the general I3 services. Individual services comprise multiple agents. Data Services deal with the product and process representation and manipulation, object-oriented approach to modeling data, and linking of component descriptions and behaviors. Interaction Services are in charge of advanced visualization of products and processes. They provide collaborative means for spatially manipulating products and processes, as well as for operator-in-the-loop simulation. Integration Services support using the system as a tool framework, collaboration and interoperation of tools, assembling collections of tools into integrated "megaprograms", and human communication through a shared electronic notebook. Finally, Application Services manage more or less static applications that perform specific roles for SBD users (e.g., animation and spreadsheet applications). These applications are mostly commercial-off-the-shelf (COTS) products.

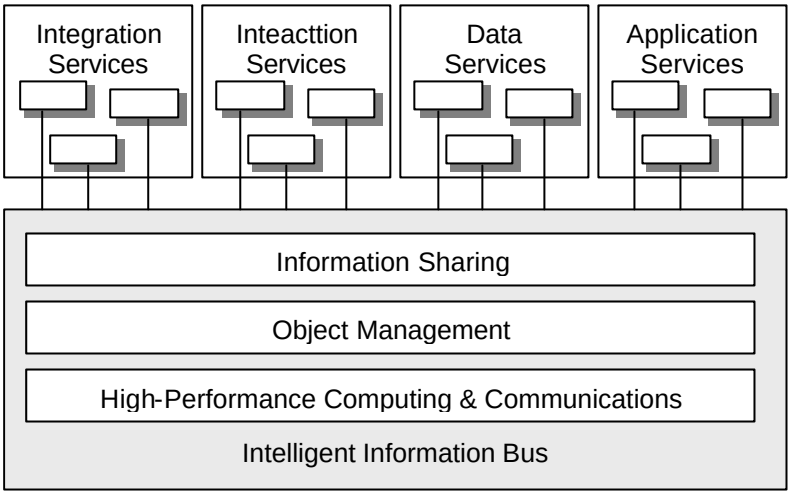

Figure 18. Architecture of the SBD system

In the lower part of Figure 18, Intelligent Information Bus provides support for communication needs of the higher level agents. In its layered architecture the Information Sharing Layer supports higher-level communications needs between entities in the system. Its duties also include publication and subscription of interest in specific objects, attributes, or events. Object Management Layer hides the complexity of communication from users and applications. High Performance Computing and Communications is the network interface layer. It isolates the underlying hardware and communications details from the Object Management Layer and higher level agents.
Incorporation of legacy codes into the SBD environment is enabled by means of wrappers. They input and output so called "smart data objects", which have associated translator methods. Smart data objects can determine the appropriate translation that is required. SBD works with a small set of interchange formats, with library of translators to convert between them.

3.3. PARKA and PARKA-DB

PARKA is a popular AI language/tool [25]. Its specific feature is its capability to scale to extremely large size applications. Also, PARKA allows for massively parallel knowledge representation.
PARKA is a frame-based language/tool. In the PARKA-based knowledge base, class, subclass, and property links are used to encode the ontology. Property values can be frames, strings, numeric values, or specialized data structures. Thus browsing a PARKA-based knowledge base on the screen is like accessing a huge semantic network. PARKA enables inferencing on knowledge bases containing millions of assertions.
PARKA-DB is another tool that uses DBMS technologies to support inferencing and data management [25]. It has significantly decreased primary memory requirements w.r.t. traditional knowledge representation systems, yet retaining inference capabilities. This is due to the integration of DBMS and KBMS technologies in PARKA-DB: DBMSs use external storage (disk) at runtime, while KBMSs enable inferencing and complex query evaluation. As a result, PARKA-DB relies primarily on the cheaper disk memory, consuming less of the more expensive internal memory at run-time.
PARKA-DB was developed to run on generic, single processor (or parallel) systems. It can process extremely complex conjunctive queries against a database. It also supports automatic knowledge discovery and data mining, in two ways:

?? verifying hypotheses against data;
?? finding relevant relationships in the database using taxonomical and other knowledge from the knowledge base.

Among the best-known applications of PARKA and PARKA-DB systems are CaPER (a case-based reasoning system), ForMAT (a case-based logistics planning system), and a set of medical information systems (see `http://www.csumd.edu/parka-db.html/` for details).

3.4. Examples of Agent Systems and Applications

From the practical perspective, intelligent agents are the most popular AI technology in recent years. Agent systems and applications are numerous, and agent-oriented software engineering opens wide the

doors to analysis and design of many software systems using intelligent agents. Amongst the application domains where agents have been applied so far are electronic commerce, Hand-held personal digital assistants (PDAs) with limited bandwidth, information gathering on the Internet, interface technology, network monitoring and control, air traffic control, business process management, industrial systems management, distributed sensing, space shuttle fault diagnosis, factory process control, and many more [26], [62]. The following systems and applications are just a few recent examples that illustrate some characteristic application domains.

DBMS-aglets. DBMS-aglets are the core of a new framework for Web-based distributed access to database systems based on Java-based mobile agents [50]. The framework's underlying idea is to avoid having a DBMS-applet at the client machine that downloads from the remote SQL server, initiates a JDBC driver, and handles a complex set of JDBC interfaces. Instead of all that, a DBMS-applet creates and fires one or more mobile agents with database-access capabilities - DBMS aglets - that migrate to the remote SQL server. There they initiate a local JDBC driver and access the database to perform any queries and other work assigned by their owner (the client program). After completing the work, DBMS aglets return to the client machine with the results. Note the full use of remote programming (Figure 7b) in this approach.

MARS-based searcher agents. Searcher agent applications typically send out mobile agents to remote Internet sites to analyze HTML pages and extract required information without having to transfer the pages over the network [8]. For example, a searcher agent can find and return the URLs of pages containing a specific keyword. In all such applications, appropriate coordination between an agent and other agents and resources in the execution environment is of fundamental importance. Mobile Agent Reactive Spaces (MARS) coordination architecture, developed at the University of Modena and Reggio Emilia, Italy, uses the so-called Linda-like coordination model. The Linda-like model is based on blackboards (shared memory spaces) for exchanging messages between the agents, empowered by organizing information in tuples and retrieving it in an associative way through a pattern-matching mechanism. As an example of MARS-based coordination between agents, consider the problem of avoiding duplicated work of different agents. Searcher agents are dynamically created and spatially autonomous, so they are unaware of other agents. Any searcher agent that arrives at a site and does some work there leaves one "marker" tuple in the tuple space. Marker tuples have the form (my_application_id, "visited"). Every other searcher agent of the same application that might subsequently arrive at the same site checks the local tuple space for the marker. If it exists, the arriving agent terminates without duplicating the work.

Mobility middleware based on mobile agents. Middleware for mobile computing must provide suitable support protocols, mechanisms, and tools for reallocating and tracing mobile users and terminals dynamically and enable communication and coordination of mobile entities [2]. Mobile agents are a convenient way to address these issues because they do not need continuous network connectivity. Connections must last only as long as it takes to inject agents from mobile terminals into the fixed network. Then the terminal can disconnect, and the agents will continue to work and will deliver the results to their users upon reconnection. Mobility middleware architectures based on mobile agents are typically organized as layered services. The lowest layer supports communication with heterogeneous distributed system (the network). The adjacent upper layer is a Java virtual machine, upon which the layer of core mobile agent services is built. It includes communication, migration, naming, security, interoperability, persistency, and QoS adaptation services. The top layer provides mobile computing services (all based on mobile agents) such as the user virtual environment (a uniform view of working environments independent of the users' current locations and specific terminals), mobile virtual terminal (preserving the terminal execution state for restoration at new locations, including active processes and subscribed services), and virtual resource management (maintaining access to resources and services).

Fuzzy intelligent agents in e-commerce. Web advertising is essential in e-commerce. Iona College's Machine Intelligence Institute, USA, develops fuzzy-logic-based intelligent agents for advertising purposes in targeted Web marketing [65]. If a commercial Web site has a collection of advertising subscribers, each one of them can provide an intelligent agent residing at that site, monitoring the current visitor's actions, and deciding about his/her possible interest in the advertiser's service. On behalf of the advertiser, the agent determines a bid to have its ad appear to the visitor. The point is that all the advertising agents receive information about the visitor simultaneously. Each agent then processes the information using its own knowledge base and then bids to have its ad appear. These bids are converted into a probability distribution that is used to determine which ad appears to the visitor. An advertiser pays the Web site not for a fixed number of units, but in proportion with the bids. In the fuzzy rules in the agents' knowledge bases, the antecedent variables (Vi) correspond to characteristics

useful in describing a site visitor. The consequent variable (U) corresponds to the appropriate bid. In the following examples of such fuzzy rules, values of fuzzy variables are shown in italics:

If age is *very young* and income is *high*, then U is u_1.
If age is *middle* and income is *low*, then U is u_5.

3.5. Examples of KDD/DM Systems and Applications

KDD/DM systems and various systems that support the KDD process are being developed intensively since mid-1990s. Table 1 shows some important application domains and typical problems that KDD/DM systems are used for in these domains [10], [20], [28], [40].

Table 1 - KDD/DM application domains and typical problems

Domain	Problem
Medicine	Discovering side-effects of drugs, genetic sequence analysis, treatment costs analysis
Finance	Credit/Loan approval, bankruptcy prediction, stock market analysis, fraud detection, unauthorized account access detection, investment selection
Social sciences	Demographic data analysis, prediction of election results, voting trends analysis
Astronomy	Analysis of satellite images
Law	Tax fraud detection, identification of stolen cars, fingerprints recognition
Marketing	Sale prediction, identification of consumer and product groups, frequent-flyer patterns
Engineering	Discovering patterns in VLSI circuit layouts, predicting aircraft-component failures
Insurance	Detection of extremely high or chained claims
Agriculture	Classification of plant diseases
Publishing	Discovering reader profiles for special issues of journals and magazines

The following brief descriptions illustrate the variety of current KDD/DM systems and applications.

Recon. Lockheed Martin Research Center has developed Recon, a KDD/DM-based stock-selection advisor [28]. It analyzes a database of over 1000 most successful companies in USA, where data are stored for each quarter from 1987 to date. For each company in the database the data change historically, and each data record in the database contains over 1000 fields, such as the stock price for that company, the trend of price and profit change, the number of analysts monitoring the company's business, etc. Recon predicts return on investment after three months of buying some stock and recommends whether to by the stock or not. The results of the analysis are patterns in the form of rules that classify stock as exceptional and non-exceptional. Recon recommends to by the stock if its return on investment can be classified as exceptional. Since the users are financial experts, not computer ones, Recon has a rich, easy-to-use graphical user interface. Pattern evaluation is done interactively, letting the users analyze and manually fine-tune the rules detected in the analysis.

CiteSeer. There are huge amounts of scientific literature on the Web. In that sense, the Web makes up a massive, noisy, disorganized, and ever-growing database. It is not easy to quickly find useful and relevant literature in such a database. CiteSeer is a custom-digital-library generator that performs information filtering and KDD functions that keep users up-to-date on relevant research [5]. It downloads Web publications in a general research area (e.g., computer architectures), creates a specific digital library, extracts features from each source in the library, and automatically discovers those publications that match the user's needs.

JAM. The growing number of credit card transactions on the Internet increases the risks of credit card numbers being stolen and subsequently used to commit fraud. The JAM (Java Agents for Metalearning) system provides distributed DM capabilities to analyze huge numbers of credit card transactions processed on the Internet each day, in search of fraudulent transactions [10]. Many more transactions are legitimate than fraudulent. JAM divides a large data set of labeled transactions (either fraudulent or legitimate) into smaller subsets and applies DM techniques to generate classifiers and subsequently generate a metaclassifier. It uses a variety of learning algorithms in generating the classifiers.

Subdue. It is possible to search graphs and discover common substructures in them. That is the basis of Subdue, the system developed to perform substructure discovery DM on various databases represented as graphs [12]. Subdue performs two key DM techniques, called unsupervised pattern discovery and supervised concept learning from examples. The system has been successfully used in discovering substructures in CAD circuit layouts, structures of some protein categories, program source code, and

aviation data. Experts in the corresponding domains ranked all of the substructures that have been discovered by Subdue highly useful.

Advanced Scout. The purpose of the KDD/DM system Advanced Scout is discovering patterns in the basketball game [4]. It helps coaches of the NBA basketball teams in analyzing their decisions brought during the games. Advanced Scout lets a coach select, filter, and transform data from a common database of all the games in the current NBA season, the database that is accessible to all NBA coaches. The database contains huge amounts of data per game, all time-stamped, and showing everything about individual players, their roles, shooting scores, blocks, rebounds, etc. The coach uses his domain knowledge (of the game and of his team) and can ask Advanced Scout DM-queries about the shooting performance, optimal playset, and so on. An example of patterns that Advance Scout discovers is "When Price was the guard, Williams' shooting score was 100%". Video records of each game facilitate interactive pattern evaluation process.

4. Some Recent Results

Due to the new research results, successful projects and systems, and current trends in knowledge modeling, a number of traditionally important AI application domains is undergoing a kind of constant revisiting in recent years. As a result, AI is getting more and more integrated with other software disciplines. The following sections briefly review some recent results in that sense.

4.1. A Framework for Building Intelligent Manufacturing Systems

Some results, techniques and trends in software engineering and AI, achieved during the 1990s, have paved the ground for the development of a systematic approach to design of software for intelligent manufacturing systems [18]. The approach is based on a multilevel, general object-oriented model of intelligent systems, called OBOA (OBject-Oriented Abstraction). It has been developed starting from some observations regarding several other current methods and software design and development tools for intelligent manufacturing systems. The other methods and tools either stress particular components of intelligence (e.g., high-level domain expertise, or learning capabilities, or fuzziness of decisions), or their domain dependence (e.g., monitoring and control systems, or CAPP systems). It is usually difficult to make extensions of such methods and tools, nor is it easy to reuse their components in developing intelligent manufacturing systems. Considerable efforts are being dedicated to the development of interoperable software components, distributed object environments, and flexible and scalable applications to overcome some of these problems.

The OBOA framework starts with a well-founded software engineering principle, making clear distinction between generic, low-level intelligent software components, and domain-dependent, high-level components of an intelligent manufacturing system. Key ideas of OBOA are illustrated in Figure 19. Each software component of an intelligent manufacturing system is defined as belonging to a certain level of abstraction. Five such levels of abstraction have been defined in OBOA so far: the *primitives* level (where components are, e.g., logical expressions, frame slots, rule clauses, and neurons), the *units* level (things like rules, frames, neural networks, fuzzy sets and their combinations), the *blocks* level (sensors, planners, controllers), the *system* level (assembly lines, robotic cells, CAPP systems), and the *integration* level (multiple intelligent agents, distributed manufacturing systems), Figure 19a. Also, components at each level can be specified along several dimensions, such as knowledge representation, reasoning, knowledge acquisition, etc., Figure 19b.

Ideally, components at each level can be built using only the components from the adjacent lower level. This means, for example, that various sensors, controllers, and planners defined at the blocks level could be specified only in terms of rules, frames, neural networks, and other lower-level components. Our experience shows that this is not always possible in practice, or it is not always optimal. In other words, drawing strict dividing lines between the levels of abstraction is not an easy task, and can lead to inflexible knowledge and problem representations. However, it is also our experience that the distribution of concepts, techniques, operations, methods, and tools over OBOA levels and dimensions, closely corresponds to that of today's intelligent manufacturing systems. It also makes the process of building intelligent manufacturing systems reusable, easily extensible, and adjustable.

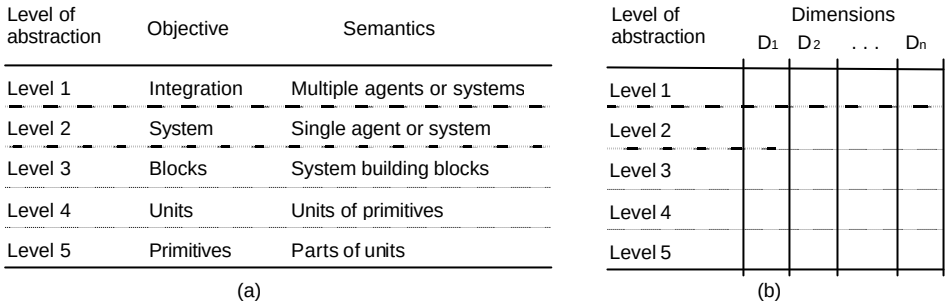

Figure 19. The OBOA model (a) levels of abstraction (b) dimensions

4.2. Towards the Ontology of Intelligent Tutoring Systems

Ontologies are nowadays a hot topic in the domain of intelligent tutoring systems (ITS). There are several approaches to development of ontologies in that domain. One of them is called GET-BITS (GEneric Tools for Building ITSs) [19]. The heart of the ITS ontology in GET-BITS is a layered, hierarchical scheme of composition of constituent parts - components and agents - of any ITS. That scheme is much like the one of the OBOA model. In GET-BITS, *primitives* are components like plain text, logical expressions, attributes and numerical values. They are used to compose *units* like rules, frames, and different utility functions. These are then used as parts of certain building *blocks* that exist in every ITS, e.g. topics, lessons and teaching strategies. At the *system* level, we have self-contained systems or agents like explanation planners, student modeling agents, and learning actors, all composed using different building blocks. Finally, at the *integration* level there are collaborative learning systems, distributed learning environments, and Web-based tutoring systems.
The integrated concept of the *ITS ontology* is split in GET-BITS into a number of separate but interrelated ontologies for all constituent parts of an ITS, such as *System ontology*, *Domain knowledge ontology*, *Pedagogical knowledge ontology*, and *Interface ontology*. These ontologies are defined in small composable modules at each level of abstraction and using the top-down approach, so that the knowledge they need can be assembled. All ontologies have the same starting point (the root concept) defined as an abstract *knowledge element*. In this way, each ontology assigned to a higher level of abstraction includes the taxonomies of the lower levels, and forms a set of inclusion lattices of ontologies.

4.3. Ontologies and Software Patterns - a Convergence?

Software patterns are solutions to recurring specific problems that arise in software analysis and design, software architectures, and organization of software projects. Such patterns provide a common vocabulary for software engineers to communicate, document, and explore engineering alternatives. They contain knowledge and experience that underlies many redesign and recoding efforts of developers that have struggled to achieve greater reuse and flexibility in their software.
Although design patterns and ontologies are not the same, recent research results show how they overlap to an extent [17]. First, both of them are about vocabularies, about knowledge, and about "architectural armatures". Both concepts also describe things at the knowledge level. Ontologies are more commonsense-oriented; software patterns are more concrete. Patterns are often about Earthly things such as software design, but can be about more abstract activities as well (e.g., organizational patterns and analysis patterns [54]). Next, it is possible to draw an analogy between libraries of ontologies and catalogues of software patterns. Although catalogues of patterns are not ready-to-use building blocks such as ontologies from the libraries, there *are* some attempts already to make them ready-to-use blocks. Also, it doesn't take a hard mental shift to view ontologies as abstract patterns, or knowledge skeletons of some domains. All these facts support the possibility of using software patterns along the other sources of ontology design [17].

5. Conclusions

Modern knowledge modeling techniques, their established practices, and hot topics in that domain stem from advancements in the domains being modeled and from the following two other sources:

?? results of developments in computer science, technology, and engineering during the last decade (such as adopted practices of object-oriented software design, layered software architectures, hybrid systems, multimedia systems, distributed systems, and the Internet);

?? research and modeling efforts in some specific fields of AI (ontologies and knowledge sharing, knowledge processing, intelligent databases, knowledge discovery, data mining, intelligent agents, virtual organizations, and the like).

An interesting observation follows from analysis of knowledge modeling in a number of projects, applications, and systems developed during the last decade: many traditionally important AI application domains have been revisited and re-modeled according to the new achievements in knowledge modeling and software engineering. Another observation is a notable increase in integration of AI with other software disciplines. That increase comes from advancements in knowledge modeling and the fact that many research and development communities have gradually realized that AI is not useful only in knowledge-based systems, but in all software systems [11]. All software systems model the world one way or another, and all of them need to model and represent some knowledge of the world.

6. References

[1] Artificial Intelligence Applications Institute, Knowledge Management, Available at: http://www.aiai.ed.ac.uk/~alm/kamlnks.html, 2001.

[2] P. Bellavista, A. Corradi, and C. Stefanelli, Mobile Agent Middleware for Mobile Computing, *IEEE Computer* 34, March (2001), 73-81.

[3] A. Bergholz, Extending Your Markup: An XML Tutorial, IEEE Internet Computing 4, July/August (2000), 74-79.

[4] I. Bhandari, E. Colet, J. Parker, Z. Pines, R. Pratap, and K. Ramanujam, Advanced Scout: data mining and knowledge discovery in NBA data, *Data Mining and Knowledge Discovery* 1 (1997), 121-125.

[5] K.D. Bollacker, S. Lawrence, and C. Lee Giles, Discovering relevant scientific literature on the Web, *IEEE Intelligent Systems* 15, March-April (2000), 42-47.

[6] P. De Bra, P. Brusilovsky, and G.-J. Houben, Adaptive Hypermedia: From Systems to Framework, *ACM Computing Surveys* 31, December (1999), 1-6.

[7] P. Brusilovsky, Adaptive and Intelligent Technologies for Web-based Education, *Künstliche Intelligenz* 4, Special Issue on Intelligent Systems and Teleteaching (1999), 19-25.

[8] G. Cabri, L. Leonardi, and F. Zambonelli, MARS: A Programmable Coordination Architecture for Mobile Agents, *IEEE Internet Computing* 4, July/August (2000), 26-35.

[9] M. Campione and K. Walrath, *The Java Tutorial - Object-Oriented Programming for the Internet*, Second Ed., Addison-Wesley, Reading, MA, 1998.

[10] P.K. Chan, W. Fan, A. Prodromidis, and S.J. Stolfo, Distributed data mining in credit card fraud detection, *IEEE Intelligent Systems* 14, November/December (1999), 67-74.

[11] B. Chandrasekaran, J.R. Josephson and V.R. Benjamins, What Are Ontologies, and Why Do We Need Them?, *IEEE Intelligent Systems* 14, January/February (1999), 20-26.

[12] D.J. Cook and L.B. Holder, Graph-based data mining, *IEEE Intelligent Systems* 15, March-April (2000), 32-41.

[13] J.H. Cook, XML Sets Stage for Efficient Knowledge Management, *IEEE IT Professional* 2, May/June (2000), 55-57.

[14] B. Czejdo, C.F. Eick and M. Taylor, Integrating Sets, Rules, and Data in an Object-Oriented Environment, *IEEE Expert* 8, February (1993), 59-66.

[15] P. Dabke, Enterprise Integration via CORBA-based Information Agents, *IEEE Internet Computing* 3, September/October (1999), 49-57.

[16] S. Decker, S. Melnik, F. Van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, and I. Horrocks, The Semantic Web: The Roles of XML and RDF, IEEE Internet Computing 4, September/October (2000), 63-74.

[17] V. Devedzic, Ontologies: Borrowing from Software Patterns, *ACM intelligence Magazine* 10, Fall (1999), 14-24.

[18] V. Devedzic and D. Radovic, A Framework for Building Intelligent Manufacturing Systems, *IEEE Transactions on Systems, Man, and Cybernetics, Part C - Applications and Reviews* 29, (1999), 422-439.
[19] V. Devedzic, D. Radovic and Lj. Jerinic, Innovative Modeling Techniques on Intelligent Tutoring Systems, in: *Knowledge-Based Paradigms: Innovative Teaching and Learning*, L.C. Jain, ed., CRC Press, Baton Rouge, USA, 2000, pp. 189-234.
[20] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, The KDD process for extracting useful knowledge from volumes of data, *Communications of The ACM* 39, November (1996), 27-34.
[21] T. Finin, Y. Labrou, and J. Mayfield, KQML as an Agent Communication Language, in: *Software Agents*, J.M. Bradshaw, ed., MIT Press, Cambridge, MA, 1997, pp. 248-260.
[22] FIPA (Foundation for Intelligent Physical Agents), *FIPA '99 Specification, Part 2: Agent Communication Language*, Available at: http://www.fipa.org, 1999.
[23] E.J. Garrity and J.C. Sipior, Multimedia as a Vehicle for Knowledge Modeling in Expert Systems, *Expert Systems with Applications* 7 (1994), 397-406.
[24] Haley Enterprise, Reasoning about Rete++, White paper available at: http://www.haley.com, October 1999.
[25] J. Hendler, K. Stoffel, M. Taylor, D. Rager and B. Kettler, *PARKA-DB: A Scalable Knowledge Representation System - Database PARKA*, Available at: http://www.csumd.edu/parka-db.html/, 1999.
[26] M. Huhns and M. Singh, eds., *Readings in Agents*, Morgan Kaufmann, San Mateo, CA, 1998.
[27] N.R. Jennings and J.R. Campos, Towards a Social Level Characterisation of Socially Responsible Agents, *IEE Proceedings on Software Engineering*, Special Issue on Agent-Based Systems, February (1997), 11-25.
[28] G.H. John, P. Miller, and R. Kerber, Stock selection using rule induction, *IEEE Expert* 11, October (1996), 52-58.
[29] E.A. Kendall, Role Modeling for Agent System Analysis, Design, and Implementation, *IEEE Concurrency* 8, April/June (2000), 34-41.
[30] N. Kock, Benefits for Virtual Organizations from Distributed Groups, *Communications of the ACM* 45, November (2000), 107-112.
[31] Y. Labrou, T. Finin, and Y. Peng, Agent Communication Languages: The Current Landscape, *IEEE Intelligent Systems* 14, March/April (1999), 45-52.
[32] C. Larman, *Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design*, Prentice-Hall, Upper Saddle River, NJ, 1998.
[33] O. Lassila and R. Swick, Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation (1999), available at http://www.w3.org/TR/REC-rdf-syntax/.
[34] G. Lawton, Knowledge Management: Ready for Prime Time?, *IEEE Computer* 34, February (2001), 12-14.
[35] D. O'Leary, Knowledge-Management Systems, *IEEE Intelligent Systems* 13, May/June (1998), 30-33.
[36] D. O'Leary, D. Kuokka and R. Plant, Artificial Intelligence and Virtual Organizations, *Communications of The ACM* 40, January (1997), 52-59.
[37] N. Lehrer et al., Key I3 Services (KIS) Working Draft, available at: http://web-ext2.darpa.mil/iso/i3/, 1999.
[38] J. Liebowitz, *Knowledge management: learning from knowledge engineering*, CRC Press, Boca Raton, FL, 2001.
[39] E. Mamdani and J. Pitt, Responsible Agent Behavior: A Distributed Computing Perspective, *IEEE Internet Computing* 4, September/October (2000), 27-31.
[40] C.J. Matheus, P.C. Chan, and G. Piatetsky-Shapiro, Systems for knowledge discovery in databases, *IEEE Transactions on Knowledge and Data Engineering* 5, 1993, 903-913.
[41] L.R. Medsker, *Hybrid Intelligent Systems*, Kluwer Academic Publishers, Amsterdam, 1994.
[42] D. Milojicic, Agent Systems and Applications, *IEEE Concurrency* 8, April/June (2000), 22-23.
[43] R. Mizoguchi, A Step Towards Ontological Engineering, in: *Proceedings of The 12th National Conference on AI of JSAI*, June 1998, pp.24-31.
[44] J.P. Muller, M.J. Wooldridge and N.R. Jennings, *Intelligent Agents*, 3 Vols, Springer Verlag, NY, 1994-1996.
[45] D.T. Ndumu an H.S. Nwana, Research and Development Challenges for Agent-Based Systems, IEE Proceedings on Software Engineering 144, February (1997), 2-10.
[46] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator and W.R. Swartout, Enabling Technology for Knowledge Sharing, *AI Magazine*, Fall (1991), 36-56.

[47] A. Newell, The Knowledge Level, *Artificial Intelligence* 18 (1982), 87-127.
[48] Object Management Group, *OMG XML Metadata Interchange (XMI) Specification*, Available at: http://www.omg.otg, 2001.
[49] J. Odell and C. Bock, Suggested UML Extensions for Agents, Response to the OMG Analysis and Design Task Force UML RTF 2.0 Request for Information, Available at: http://www.omg.org, 1999.
[50] S. Papastavrou, G. Samaras, and E. Pitoura, Mobile Agents for World Wide Web Distributed Database Access, IEEE Transactions on Knowledge and Data Engineering 12 September/October (2000), 802-820.
[51] K. Parsaye and M. Chignell, Intelligent Databases: Object-Oriented, Deductive Hypermedia Technologies, John Wiley, NY, 1993.
[52] J. Roy and A. Ramanujan, XML: Data's Universal Language, *IEEE IT Professional* 2, May/June (2000), 32-36.
[53] S. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach*, Prentice-Hall, Englewood Cliffs, NJ, 1995.
[54] D. Schmidt, M. Fayad and R.E. Johnson, Software Patterns, *Communications of The ACM* 39, October (1996), 37-39.
[55] J. Self, The defining characteristics of intelligent tutoring systems research: ITSs care, precisely, *International Journal of Artificial Intelligence in Education* 10 (1999), 1-17.
[56] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, Englewood Cliffs, NJ, 1996.
[57] Y. Shoham, Agent-oriented programming, *Artificial Intelligence* 60 (1993), 51–92.
[58] R. Sison and M. Shimura, Student Modeling and Machine Learning, *International Journal of Artificial Intelligence in Education* 9 (1998), 128-158.
[59] W. Swartout and A. Tate, Ontologies, *IEEE Intelligent Systems* 14, January/February (1999), 18-19.
[60] P.T. Wojciechowski and P.Sewell, Nomadic Pict: Language and Infrastructure Design for Mobile Agents, *IEEE Concurrency* 8, April/June (2000), 42-52.
[61] D. Wong, N. Paciorek, and D. Moore, Java-based Mobile Agents, *Communications of the ACM* 42, March (1999), 92-102.
[62] M. Wooldridge, *Reasoning about Rational Agents*, MIT Press, Cambridge, MA, 2000.
[63] M. Wooldridge, Intelligent Agents, in: *Multiagent Systems*, G. Weiss, ed., MIT Press, Cambridge, MA, 1999, pp. 3-51.
[64] M.J. Wooldridge and N.R. Jennings, Software Engineering with Agents: Pitfalls and Pratfalls, *IEEE Internet Computing* 3, May/June (1999), 20-27.
[65] R.R. Yager, Targeted E-commerce Marketing Using Fuzzy Intelligent Agents, *IEEE Intelligent Systems* 15, November/December (2000), 42-45.