Sponsored by the Office of Naval Research (ONR)

ONR Decision-Support Workshop Series

Continuing the Revolution in Military Affairs (RMA)

hosted by the

Collaborative Agent Design Research Center (CADRC)
Cal Poly State University, San Luis Obispo, CA

in conjunction with

CDM Technologies, Inc.
San Luis Obispo, CA

Proceedings of Workshop held on June 5-7, 2001

at
The Clubs at Quantico, Quantico Marine Base
Quantico, VA

September, 2001

Information-Centric Decision-Support Systems: A Blueprint for '*Interoperability*'

Jens Pohl, Ph.D.
Collaborative Agent Design Research Center
Cal Poly, San Luis Obispo, CA, USA

For the past 20 years the US military services have suffered under the limitations of stove-piped computer software applications that function as discrete entities within a fragmented data-processing environment. Lack of interoperability has been identified by numerous think tanks, advisory boards, and studies, as the primary information systems problem (e.g., Army Science Board 2000, Air Force SAB 2000 Command and Control Study, and NSB Network-Centric Naval Forces 2000). Yet, despite this level of attention, all attempts to achieve ***interoperability*** within the current ***data-centric*** information systems environment have proven to be expensive, unreliable, and generally unsuccessful.

The Apparently Elusive Goal of '*Interoperability*'

The expectations of true interoperability are threefold. First, interoperable applications should be able to integrate related functional sequences in a seamless and user transparent manner. Second, this level of integration assumes the sharing of ***information*** from one application to another, so that the results of the functional sequence are automatically available and similarly interpreted by the other application. And third, any of the applications should be able to enter or exit the integrated interoperable environment without jeopardizing the continued operation of the other applications. These conditions simply cannot be achieved by computer software that processes numbers and meaningless text with predetermined algorithmic solutions through hard-coded dumb data links.

Past approaches to interoperability have basically fallen into three categories. Attempts to create common architectures have largely failed because this approach essentially requires existing systems to be re-implemented in the common (i.e., new) architecture. Attempts to create bridges between applications within a confederation of linked systems have been faced with three major obstacles. First, the large number of bridges required (i.e., the square of the number of applications). Second, the fragility associated with hard-coded inter-system data linkages. Third, the cost of maintaining such linkages in a continuously evolving information systems environment. The third category of approaches has focused on achieving interoperability at the interface boundary. For anything other than limited presentation and visualization capabilities, this approach cannot accommodate dynamic data flows, let alone constant changes at the more useful information level.

These obstacles to interoperability and integration are largely overcome in an information-centric software systems environment by embedding in the software some understanding of the information being processed. How is this possible? Surely computers cannot be expected to understand anything. Aren't they just dumb electronic machines that simply execute programmed instructions without any regard to what either the instructions, or the information to which the instructions apply, mean? The answer is no, it is all a matter of ***representation*** (i.e., how the information is structured in the computer).

The Notion of '*Information-Centric*'

The term ***information-centric*** refers to the representation of information in the computer, not to the way it is actually stored in a digital machine. This distinction between ***representation*** and ***storage*** is important, and relevant far beyond the realm of computers. When we write a note with a pencil on a sheet of paper, the content (i.e., meaning) of the note is unrelated to the storage device. A sheet of paper is designed to be a very efficient storage medium that can be easily stacked in sets of hundreds, filed in folders, bound into volumes, folded, and so on. However, all of this is unrelated to the content of the written note on the paper. This content represents the meaning of the sheet of paper. It constitutes the purpose of the paper and governs what we do with the sheet of paper (i.e., its use). In other words, the nature and efficiency of the storage medium is more often than not unrelated to the content or representation that is stored in the medium.

In the same sense, the way in which we store bits (i.e., 0s and 1s) in a digital computer is unrelated to the meaning of what we have stored. When computers first became available they were exploited for their fast, repetitive computational capabilities and their enormous storage capacity. Application software development progressed rapidly in a ***data-centric*** environment. Content was stored as data that were fed into algorithms to produce solutions to predefined problems in a static problem solving context. It is surprising that such a simplistic and artificially contrived problem solving environment was found to be acceptable for several decades of intensive computer technology development.

When we established the Collaborative Agent Design Research Center at Cal Poly in 1986, we had a vision. We envisioned that users should be able to sit down at a computer terminal and solve problems collaboratively with the computer. The computer should be able to continuously assist and advise the user during the decision-making process. Moreover, we postulated that one should be able to develop software modules that could spontaneously react in near real-time to changing events in the problem situation, analyze the impact of the events, propose alternative courses of action, and evaluate the merits of such proposals. What we soon discovered, as we naively set out to develop an intelligent decision-support system, is that we could not make much headway with ***data*** in a dynamically changing problem environment.

Initially focusing on engineering design, we had no difficulties at all developing a software module that could calculate the daylight available inside a room, as long as we specified to the computer the precise location and dimensions of the window, the geometry of the room, and made some assumptions about external conditions. However, it did not seem possible for the computer to determine on its own that there was a need for a window and where that window might be best located. The ability of the computer to make these determinations was paramount to us. We wanted the computer to be a useful assistant that we could collaborate with as we explored alternative design solutions. In short, we wanted the computer to function ***intelligently*** in a dynamic environment, continuously looking for opportunities to assist, suggest, evaluate, and, in particular, alert us whenever we pursued solution alternatives that were essentially not practical or even feasible.

We soon realized that to function in this role our software modules had to be able to ***reason***. However, to be able to reason the computer needs to have something akin to ***understanding*** of the ***context*** within which it is supposed to reason. The human cognitive system builds context from knowledge and experience using ***information*** (i.e., data with attributes and relationships) as its basic building block. Interestingly enough the storage medium of the information, knowledge

and context held by the human brain is billions of neurons and trillions of connections (i.e., synapses) among these neurons that are as unrelated to each other as a pencilled note and the sheet of paper on which it is stored.

What gives meaning to the written note is its ***representation*** within the framework of a language (e.g., English) that can be understood by the reader. Similarly, in a computer we can establish the notion of ***meaning*** if the stored data are represented in an ontological framework of objects, their characteristics, and their interrelationships. How these objects, characteristics and relationships are actually stored at the lowest level of bits (i.e., 0s and 1s) in the computer is immaterial to the ability of the computer to undertake reasoning tasks. The conversion of these bits into data and the transformation of data into information, knowledge and context takes place at higher levels, and is ultimately made possible by the skillful construction of a network of richly described objects and their relationships that represent those physical and conceptual aspects of the real world that the computer is required to reason about.

This is what is meant by an information-centric computer-based decision-support environment. One can further argue that to refer to the ability of computers to ***understand*** and ***reason*** about ***information*** is no more or less of a trick of our imagination than to refer to the ability of human beings to understand and reason about information. In other words, the countless minuscule charges that are stored in the neurons of the human nervous system are no closer to the representation of information than the bits (i.e., 0s and 1s) that are stored in a digital computer. However, whereas the human cognitive system automatically converts this collection of charges into information and knowledge, in the computer we have to construct the framework and mechanism for this conversion Such a framework of objects, attributes and relationships provides a system of integrated software applications with a common language that allows software modules (now popularly referred to as ***agents***) to ***reason*** about events, monitor changes in the problem situation, and collaborate with each other as they actively assist the user(s) during the decision-making process. One can say that this ***ontological framework*** is a virtual representation of the real world problem domain, and that the agents are dynamic tools capable of pursuing objectives, extracting and applying knowledge, communicating, and collaboratively assisting the user(s) in the solution of current and future real world problems.

Definitions: Data, Information, and Knowledge

It is often lamented that we human beings are suffering from an ***information overload***. This is a myth, as shown in Fig.1 there is no information overload. Instead we are suffering from a data overload. The confusion between data and information is not readily apparent and requires further explanation. Unorganized data are voluminous but of very little value. Over the past 15 years, industry and commerce have made significant efforts to rearrange this unorganized data into purposeful data, utilizing various kinds of database management systems. However, even in this organized form, we are still dealing with data and not information.

Data are defined as numbers and words without relationships. In reference to Fig.2, the words “town”, “dog”, “Tuesday”, “rain”, “inches”, and “min”, have little if any meaning without relationships. However, linked together in the sentence: "On Tuesday, 8 inches of rain fell in 10 min."; they become information. If we then add the context of a particular geographical region, pertinent historical climatic records, and some specific hydrological information relating to soil conditions and behavior, we could perhaps infer that: "Rainfall of such magnitude is likely to cause flooding and landslides." This becomes knowledge.

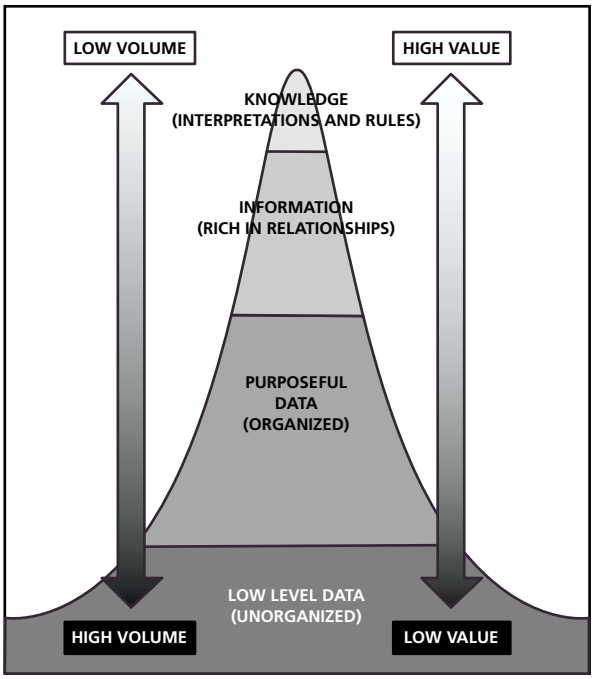

Fig.1: The ***information overload*** myth

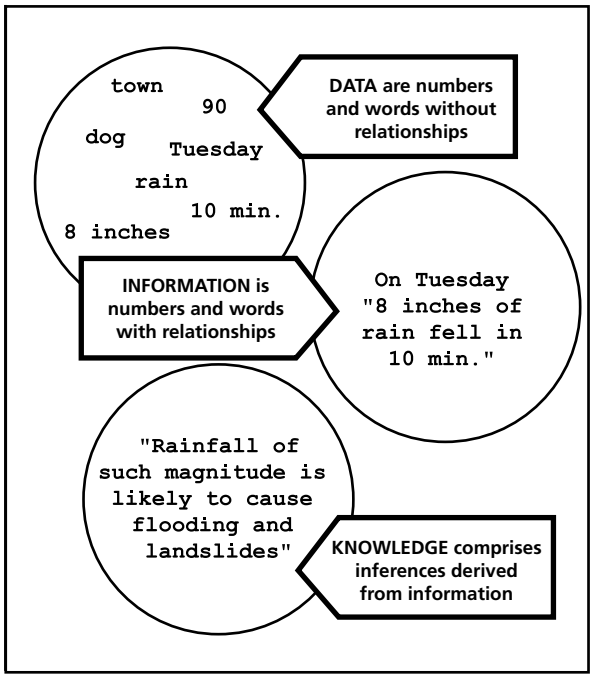

Fig.2: Data, information and knowledge

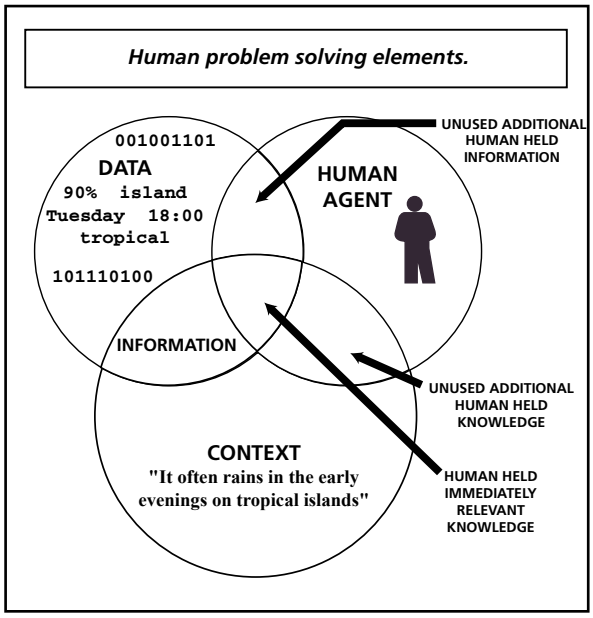

Fig.3: Unassisted problem solving

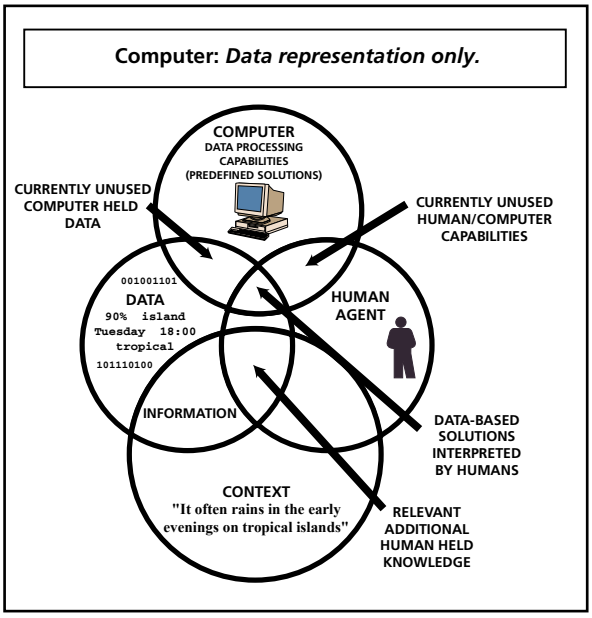

Fig.4: Limited data-processing assistance

Context is normally associated solely with human cognitive capabilities. Prior to the advent of computers, it was entirely up to the human agent to convert data into information and to infer knowledge through the addition of context. However, the human cognitive system performs this function subconsciously (i.e., automatically); therefore, prior to the advent of computers, the difference between data and information was an academic question that had little practical

significance in the real world of day-to-day activities. As shown in Fig.3, the intersection of the data, human agent and context realms provides a segment of immediately relevant knowledge.

The Data-Centric Evolution of Computer Software

When computers entered on the scene, they were first used exclusively for processing data. In fact, even in the 1980s computer centers were commonly referred to as data-processing centers. It can be seen in Fig.4 that the context realm remained outside the computer realm. Therefore, the availability of computers did not change the need for the human agent to interpret data into information and infer knowledge through the application of context. The relegation of computers to data-processing tasks is the underlying reason why even today, as we enter the 21st Century, computers are still utilized in only a very limited decision-support role. As shown in Fig.5, in this limited computer-assistance environment human decision makers typically collaborate with each other utilizing all available communication modes (e.g., telephone, FAX, e-mail, letters, face-to-face meetings). Virtually every human agent utilizes a personal computer to assist in various computational tasks. While these computers have some data sharing capabilities in a networked environment, they cannot directly collaborate with each other to assist the human decision makers in the performance of decision-making tasks. Each computer is typically limited to providing relatively low-level data-processing assistance to its owner. The interpretation of data, the inferencing of knowledge, and the collaborative teamwork that is required in complex decision-making situations remains the exclusive province of the human agents. In other words, without access to information and at least some limited context, the computer cannot participate in a distributed collaborative problem-solving arena.

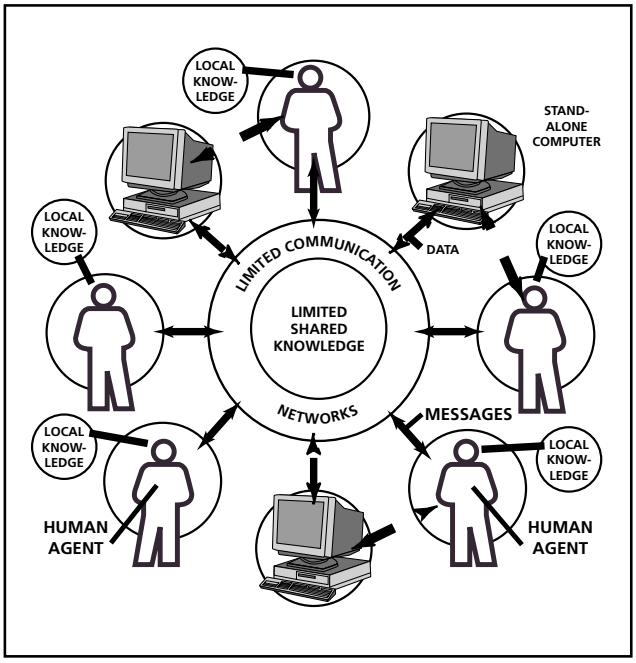

Fig.5: Limited computer assistance

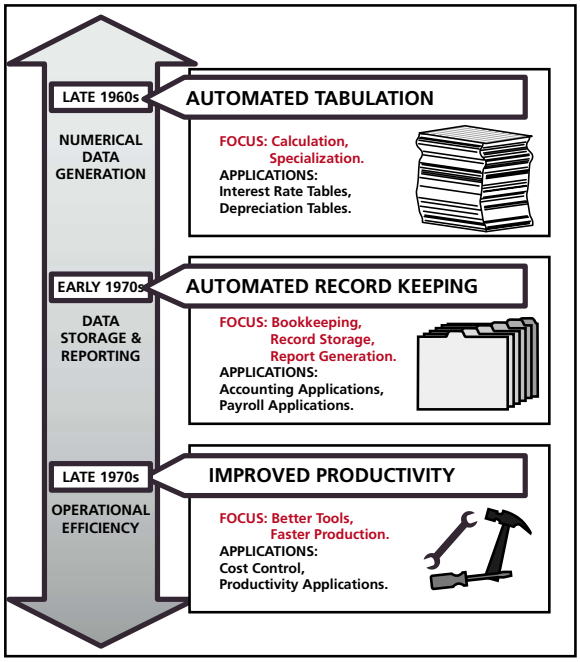

Fig.6: Evolution of business intelligence (A)

In this regard, it is of interest to briefly trace the historical influence of evolving computer capabilities on business processes and organizational structures. When the computer first became more widely available as an affordable computational device in the late 1960s, it was

applied immediately to specialized numerical calculation tasks such as interest rate tables and depreciation tables (Fig.6). During the early 1970s, these computational tasks broadened to encompass bookkeeping, record storage, and report generation. Tedious business management functions were taken over by computer-based accounting and payroll applications. By the late 1970s, the focus turned to improving productivity using the computer as an improved automation tool to increase and monitor operational efficiency.

In the early 1980s (Fig.7), the business world had gained sufficient confidence in the reliability, persistence, and continued development of computer technology to consider computers to be a permanent and powerful data-processing tool. Accordingly, businesses were willing to reorganize their work flow as a consequence of the functional integration of the computer. More comprehensive office management applications led to the restructuring of the work flow.

By the late 1980s, this had led to a wholesale re-engineering of the organizational structure of many businesses with the objective of simplifying, streamlining, and downsizing. It became clear that many functional positions and some entire departments could be eliminated and replaced by integrated office automation systems. During the early 1990s, the problems associated with massive unorganized data storage became apparent, and with the availability of much improved database management systems, data were organized into mostly relational databases. This marked the beginning of ordered-data archiving and held out the promise of access to any past or current data and reporting capabilities in whatever form management desired.

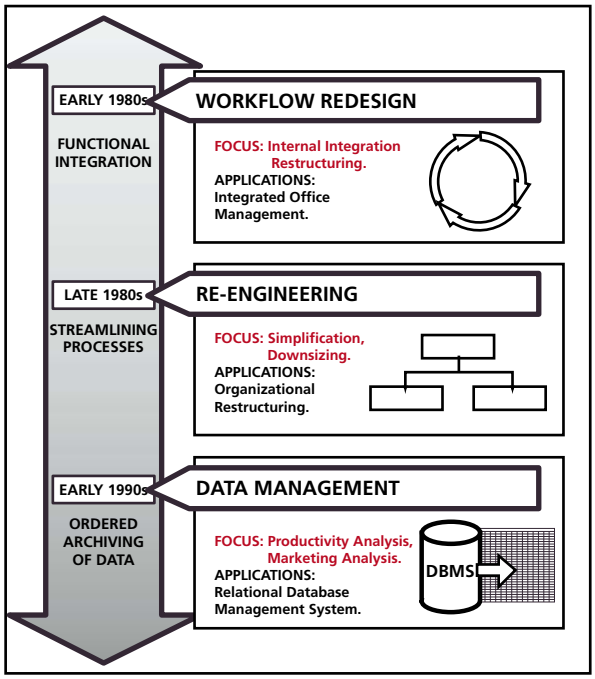

Fig.7: Evolution of business intelligence (B)

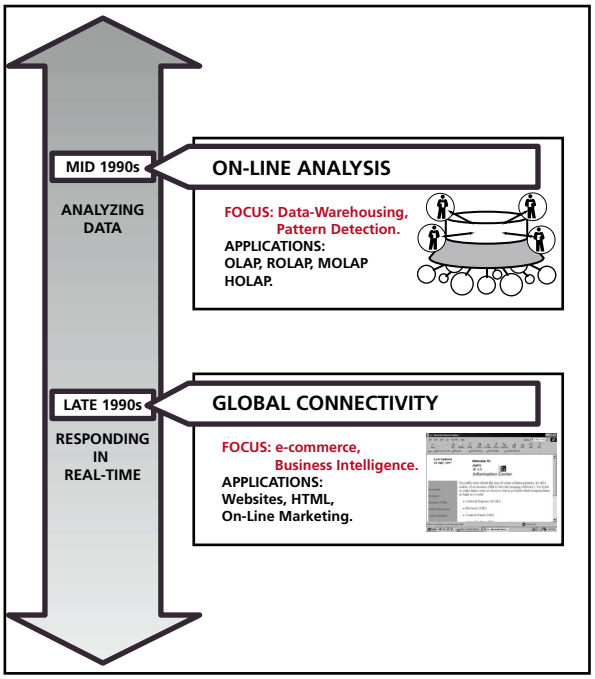

Fig.8: Evolution of business intelligence (C)

However, by the mid 1990s (Fig.8), the quickening pace of business in the light of greater competition increased the need for a higher level of data analysis, faster response, and more accurate pattern detection capabilities. During this period, the concepts of data-warehouses, data-marts, and On-Line Analytical Processing (OLAP) tools were conceived and rapidly

implemented (Humphries et al. 1999). Since then, the term 'business intelligence' has been freely used to describe a need for the continuous monitoring of business trends, market share, and customer preferences.

In the late 1990s, the survival pressure on business increased with the need for real-time responsiveness in an Internet-based global e-commerce environment. By the end of the 20th Century, business began to seriously suffer from the limitations of a data-processing environment. The e-commerce environment presented attractive opportunities for collecting customer profiles for the implementation of on-line marketing strategies with enormous revenue potential. However, the expectations for automatically extracting useful information from low-level data could not be satisfied by the methods available. These methods ranged from relatively simple keyword and thematic indexing procedures to more complex language-processing tools utilizing statistical and heuristic approaches (Denis 2000, Verity 1997).

The major obstacle confronted by all of these information-extraction approaches is the unavailability of adequate context (Pedersen and Bruce 1998). As shown previously in Fig.4, a computer-based ***data-processing*** environment does not allow for the representation of context. Therefore, in such an environment, it is left largely to the human user to interpret the data elements that are processed by the computer.

Methods for representing information and knowledge in a computer have been a subject of research for the past 40 years, particularly in the field of 'artificial intelligence' (Ginsberg 1993). However, these studies were mostly focussed on narrow application domains and did not generate wide-spread interest even in computer science circles. For example even today, at the beginning of the 21st Century, it is difficult to find an undergraduate computer science degree program in the US that offers a core curriculum class dealing predominantly with the representation of information in a computer.

The Representation of '*Context*' in a Computer

Conceptually, to represent information in a computer, it is necessary to move the context circle in Fig.4 upward into the realm of the computer (Fig.9). This allows data to enter the computer in a contextual framework, as information. The intersection of the data, context, and human agent circles provide areas in which information and knowledge are held in the computer. The prevailing approach for the practical implementation of the conceptual diagram shown in Fig.9 is briefly outlined below. As discussed earlier (Fig.2), the principal elements of information are data and relationships. We know how data can be represented in the computer but how can the relationships be represented? The most useful approach available today is to define an ontology of the particular application domain in the form of an object model. This requires the identification of the objects (i.e., elements) that play a role in the domain and the relationships among these objects (Fig.10). Each object, whether physical (e.g., car, person, building, etc.) or conceptual (e.g., event, privacy, security, etc.) is first described in terms of its behavioral characteristics. For example, a ***car*** is a kind of ***land conveyance***. As a child object of the ***land conveyance*** object, it automatically inherits all of the characteristics of the former and adds some more specialized characteristics of its own (Fig.11). Similarly, a ***land conveyance*** is a kind of ***conveyance*** and therefore inherits all of the characteristics of the latter. This powerful notion of inheritance is well supported by object-oriented computer languages such as C++ (Stroustrup 1987) and Java (Horstmann and Cornell 1999) that support the mainstream of applications software development today.

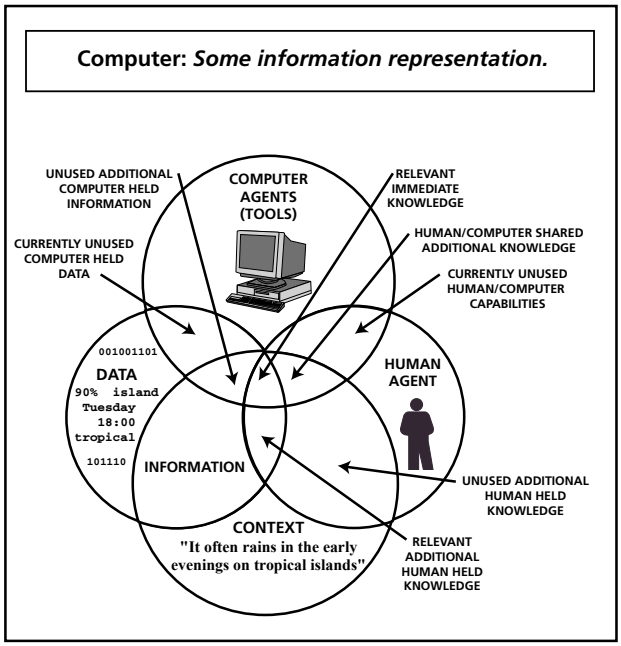

Fig.9: Early human-computer partnership

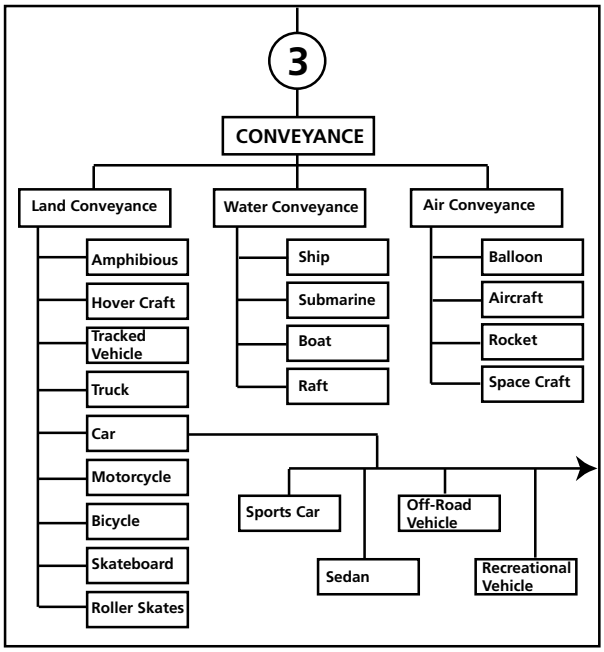

Fig.10: Branch of a typical object model

However, even more important than the characteristics of objects and the notion of inheritance are the relationships that exist between objects. As shown in Fig.12, a car incorporates many components that are in themselves objects. For example, cars typically have engines, steering systems, electric power units, and brake systems. They utilize fuel and often have an air-conditioning system.

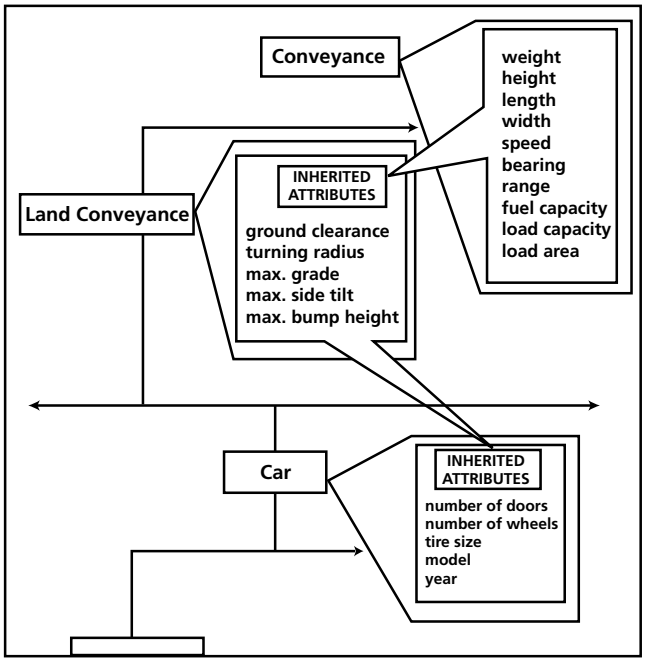

Fig.11: Object model - ***inheritance***

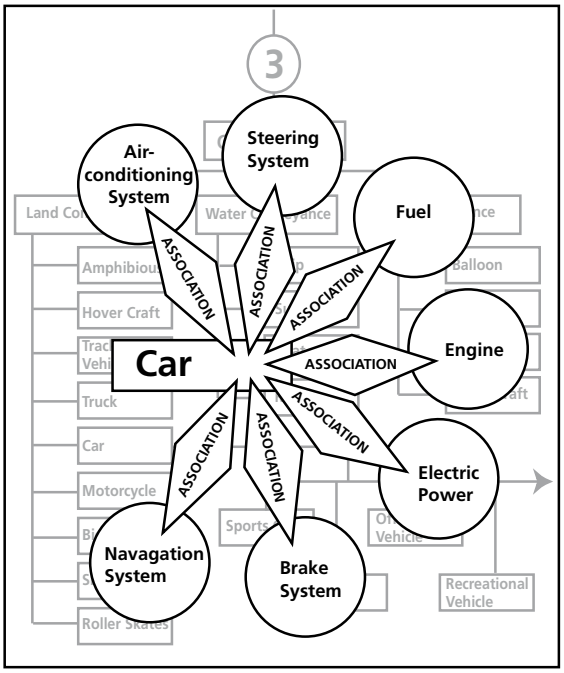

Fig.12: Object model - ***associations***

For several reasons, it is advantageous to treat these components as objects in their own right rather than as attributes of the car object. First, they may warrant further subdivision into parent and child objects. For example, there are several kinds of air-conditioning systems, just as there are several kinds of cars. Second, an air-conditioning system may have associations of its own to other component systems such as a temperature control unit, a refrigeration unit, an air distribution system, and so on. Third, by treating these components as separate objects we are able to describe them in much greater detail than if they were simply attributes of another object. Finally, any changes in these objects are automatically reflected in any other objects that are associated with them. For example, during its lifetime, a car may have its air-conditioning system replaced with another kind of air handling unit. Instead of having to change the attributes of the car, we simply delete the association to the old unit and add an association to the new unit. This procedure is particularly convenient when we are dealing with the association of one object to many objects, such as the wholesale replacement of a cassette tape player with a new compact disk player model in many cars, and so on.

The way in which the construction of such an ontology leads to the representation of information (rather than data) in a digital computer is described in Fig.13, as follows. By international agreement, the American Standard Code for Information Interchange (ASCII) provides a simple binary (i.e., digital) code for representing numbers, alphabetic characters, and many other symbols (e.g., +, -, =, ( ), etc.) as a set of ***0*** and ***1*** digits. This allows us to represent sets of characters such as the sentence ***"Police car crossing bridge at Grand Junction."*** in the computer. However, in the absence of an ontology, the computer stores this set of characters as a meaningless text string (i.e., data). In other words, in the ***data-centric*** realm the computer has no understanding at all of the meaning of this sentence. As discussed previously, this is unfortunately the state of e-mail today. While e-mail has become a very convenient, inexpensive, and valuable form of global communication, it depends entirely on the human interpretation of each e-mail message by both the sender and the receiver.

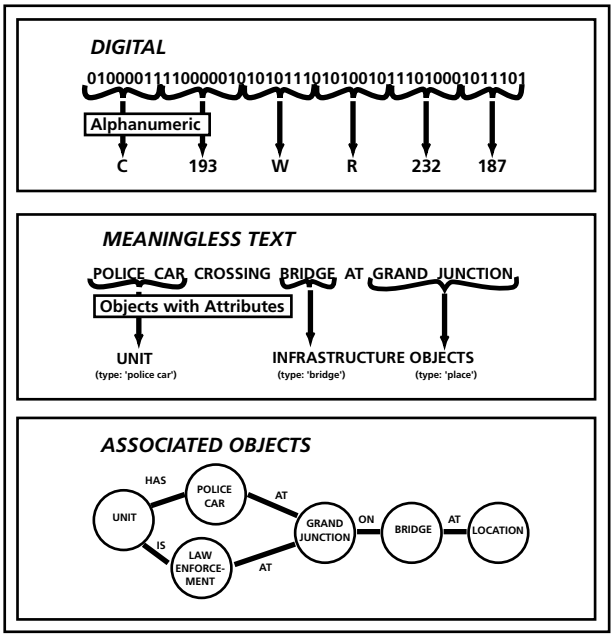

Fig.13: From ***digital*** to ***information***

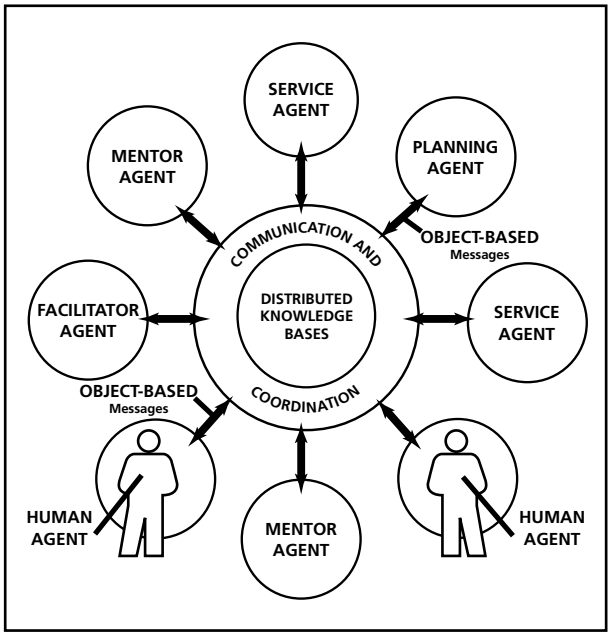

Fig.14: Types of agents

Now, if the ***"Police car crossing bridge at Grand Junction."*** message had been sent to us as a set of related objects, as shown at the bottom of Fig.13, then it should be a relatively simple matter to program computer-based agents to reason about the content of this message and perform actions on the basis of even this limited level of understanding. How was this understanding achieved? In reference to Fig.13, the police car is interpreted by the computer as an instance of a ***car*** object which is associated with a ***civilian organization*** object of kind ***police***. The ***car*** object automatically inherits all of the attributes of its parent object, ***land conveyance***, which in turn inherits all of the attributes of its own parent object, ***conveyance***. The ***car*** object is also associated with an instance of the infrastructure object, ***bridge***, which in turn is associated with a ***place*** object, ***Grand Junction***, giving it a geographical location. Even though this interpretational structure may appear primitive to us human beings, it is adequate to serve as the basis of useful reasoning and task performance by computer-based agents.

The Popular Notion of '*Intelligent Agents*'

Agents that are capable of ***reasoning*** about events, in the kind of ontological framework of ***information*** described above, are little more than software modules that can process objects, recognize their behavioral characteristics (i.e., attributes of the type shown for the objects in Fig.11), and trace their relationships to other objects. It follows, that perhaps the most elementary definition of ***agents*** is simply: "Software code that is capable of ***communicating*** with other entities to facilitate some action". Of course this communication and action capability alone does not warrant the label of ***intelligent***.

The use of the word intelligent is more confusing than useful. As human beings we tend to judge most everything in the world around us in our image. And, in particular, we are rather sensitive about the prospect of ascribing intelligence to anything that is not related to the human species, let alone an electronic machine. Looking beyond this rather emotional viewpoint, one could argue that there are levels of intelligence. At the most elementary level, intelligence is the ability to remember. A much higher level of intelligence is creativity (i.e., the ability to create new knowledge). In between these two extremes are multiple levels of increasingly intelligent capabilities. Certainly computers can remember, because they can store an almost unlimited volume of data and can be programmed to retrieve any part of that data. Whether, computers can interpret what they remember depends on how the data are represented (i.e., structured) in the software.

In this regard, the notion of ***intelligent agents*** refers to the existence of a common language (i.e., the ontological framework of information described earlier) and the ability to ***reason*** about the object characteristics and relationships embodied in the informational structure. Increasing levels of intelligent behavior can be achieved by software agents if they have access to existing knowledge, are able to act on their own initiative, collaborate with other agents to accomplish goals, and use local information to manage local resources.

Such agents may be programmed in many ways to serve different purposes (Fig.14). Mentor agents may be designed to serve as guardian angels to look after the welfare and represent the interests of particular objects in the underlying ontology. For example, a mentor agent may simply monitor the fuel consumption of a car or perform more complex tasks such as helping a tourist driver to find a particular hotel in an unfamiliar city, or alert a platoon of soldiers to a hostile intrusion within a specified radius of their current position in the battlefield (Pohl et al. 1999). Service agents may perform expert advisory tasks on the request of human users or other

agents. For example, a computer-based daylighting consultant can assist an architect during the design of a building (Pohl et al. 1989) or a Trim and Stability agent may continuously monitor the trim of a cargo ship while the human cargo specialist develops the load plan of the ship (Pohl et al. 1997). At the same time, Planning agents can utilize the results of tasks performed by Service and Mentor agents to devise alternative courses of action or project the likely outcome of particular strategies. Facilitator agents can monitor the information exchanged among agents and detect apparent conflicts (Pohl 1996). Once such a Facilitator agent has detected a potential non-convergence condition involving two or more agents, it can apply one of several relatively straightforward procedures for promoting consensus, or it may simply notify the user of the conflict situation and explain the nature of the disagreement.

An Information-Centric Transition Architecture

An information-centric decision-support system typically consists of components (or modules) that exist as clients to an integrated collection of services. Incorporating such services, the information-serving collaboration facility (Fig.15) communicates to its clients in terms of the real world objects and relationships that are represented in the information structure (i.e., the underlying ontology). The software code of each client includes a version of the ontology, serving as the common language that allows clients to communicate ***information*** rather than data.

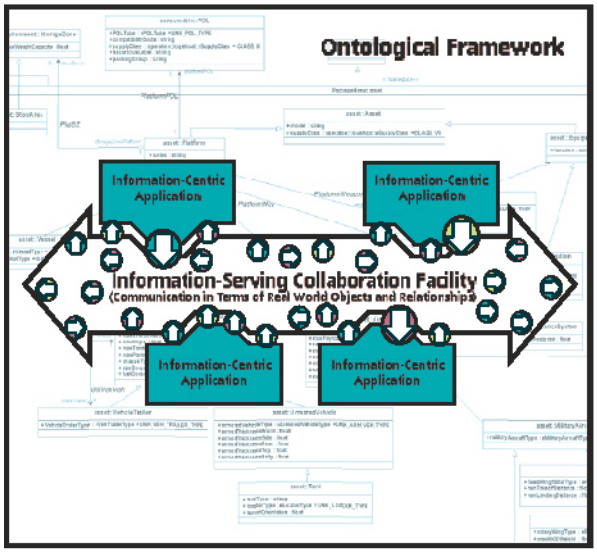

Fig.15: Information-centric interoperability.

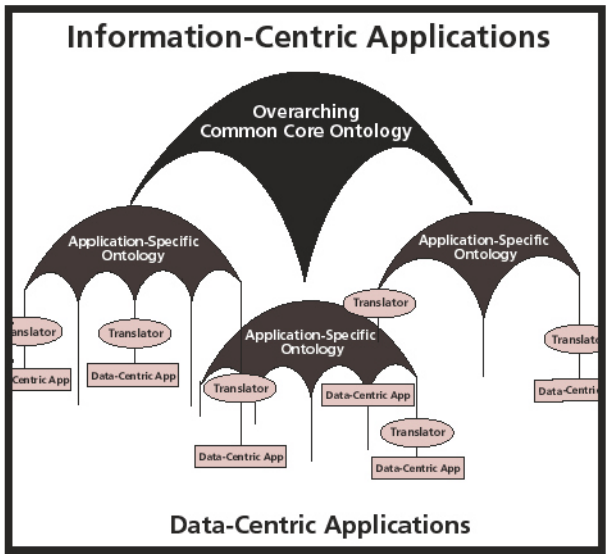

Fig.16: Transitioning to an information-centric architecture.

To reduce the amount of work (i.e., computation) that the computer has to accomplish and to minimize the volume of information that has to be transmitted within the system, two strategies can be readily implemented. First, each client can register a standing request with the collaboration facility for the kind of information that it would like to receive. This is referred to as a subscription profile, and the client has the ability to change this profile dynamically during execution if it sees cause to ask for additional or different information. For example, after receiving certain information through its existing subscription profile, a Mentor agent representing a squad of Marines may decide to request information relating to engagement

events in a different sector of the battlefield, henceforth. By allowing information to be automatically ***pushed*** to clients, the subscription service obviates the need for database queries and thereby greatly reduces the amount of work the computer has to perform. Of course, a separate query service is also usually provided so that a client can make one-time requests for information that is not required on a continuous basis.

The second strategy relates directly to the volume of information that is required to be transmitted within the system. Since the software code of each client includes a version of the ontology (i.e., common language) only the changes in information need to be communicated. For example, a Mentor agent that is watching over a squad of Marines may have more than 100 objects included in its subscription profile. One set of these objects represents an enemy unit and its warfighting capabilities. If this unit changes its position then in reality only one attribute (i.e., the location attribute) of one object may have changed. Only the changed value of this single object needs to be transmitted to the Mentor agent, since as a client to the collaboration facility it already has all of the information that has not changed.

How does this ***interoperability*** between the collaboration facility and its clients translate into a similar interoperability among multiple software applications (i.e., separate programs dealing with functional sequences in related domains)? For example, more specifically, how can we achieve interoperability between a tactical command and control system such as IMMACCS (Pohl et al. 1999) and a logistical command and control system such as SEAWAY (Wood et al. 2000)?

Since both of these software systems are implemented in an information-centric architecture, the underlying information representation can be structured in levels (Fig.16). At the highest level we define notions, concepts and object types in general terms. This overarching common core ontology sits on top of any number of lower level application specific ontologies that address the specific bias and level of granularity of the particular application domain. For example, in the core ontology an 'aircraft' may be defined in terms of its physical nature and those capabilities that are essentially independent of its role in a particular application domain. In the tactical domain this general description (i.e., representation) of an 'aircraft' is further refined and biased toward a warfighting role. In other words, the IMMACCS application sees an aircraft as an airborne weapon with certain strike capabilities. SEAWAY, on the other hand, sees an aircraft as an airborne mobile warehouse capable of transporting supplies from one point to another.

The interoperability capabilities of an information-centric software environment will also allow agents in one application to notify agents in other applications of events occurring in multiple domains. For example, the Engagement Agent in the tactical IMMACCS application is able to advise appropriate agents in the logistical SEAWAY application whenever a Supply Point ashore is threatened by enemy activity. This may result in the timely rescheduling or redirection of a planned re-supply mission. The agents are able to communicate across multiple applications at the information level through the common language of the ontological framework. Similarly, the SEAWAY application is able to rely on the ICODES (Pohl et al. 1997) ship load planning application to maintain in-transit cargo visibility, down to the location of a specific supply item in a particular container on-board a ship en-route to the sea base.

One might argue that this is all very well for newly developed applications that are by design implemented in an ***information-centric*** architecture, but what about the many existing ***data-centric*** applications that all perform strategic and indispensable functions? These existing legacy applications constitute an enormous investment that cannot be discarded overnight, for several

reasons. First, they perform critical functions. Second, it will take time to cater for these functions in the new decision-support environment. Third, at least some of these functions will be substantially modified or eliminated as the information-centric environment evolves.

As shown in Fig.16, data-centric applications can communicate with information-centric systems through translators. The function of these translators is to map those portions of the low level data representation of the external application that are important to the decision-making context, to the ontology of the information-centric system. Conversely, the same translator must be capable of extracting necessary data items from the information context and feed these back to the data-centric application. Typically, as in the case of IMMACCS (Pohl et al. 1999), this translation capability is implemented as a universal translator that can be customized to a particular external application. The translator itself, exists as a client to the information-serving collaboration facility (Fig.15) of the information-centric system and therefore includes in its software code a version of the ontology that describes the common language of that system.

Conclusion

While the capabilities of present day computer-based agent systems are certainly a major advancement over data-processing systems, we are only at the threshold of a paradigm shift of major proportions. Over the next several decades, the context circle shown in Fig.17 will progressively move upward into the computer domain, increasing the sector of "relevant immediate knowledge" shared at the intersection of the human, computer, data, and context domains. Returning to the historical evolution of business intelligence described previously in reference to Figs. 6, 7 and 8, the focus in the early 2000s will be on information management as opposed to data-processing (Fig.18). Increasingly, businesses will insist on capturing data as information through the development of business enterprise ontologies and leverage scarce human resources with multi-agent software capable of performing useful analysis and pattern-detection tasks.

An increasing number of commercial companies are starting to take advantage of the higher level collaborative assistance capabilities of computers to improve their competitive edge and overcome potential customer service difficulties. A good example is the timely detection of the fraudulent use of telephone credit card numbers. Telephone companies deal with several million calls each day, far too many for monitoring by human detectives. Instead, they have implemented intelligent computer software modules that monitor certain information relating to telephone calls and relate that information to the historical records of individual telephone users. The key to this capability is that telephone call data such as time-of-day, length of call, origin of call, and destination are stored in the computer as an information structure containing data objects, relationships, and some attributes for each data object. For example, the data ‘Columbia’ may have the attributes international, South America, uncommon telephone call destination, attached to it. In addition, relationships are established dynamically between ‘Columbia’ the telephone number of the caller, the telephone number being called, the time-of-day of the call, and so on. The result is a network of objects with attributes and relationships that is very different from the data stored in a typical commercial data-mart. This network constitutes information (rather than data) and allows hundreds of software ***agents*** to monitor telephone connections and detect apparent anomalies. What is particularly attractive about this fairly straightforward application of ***information-centric*** technology, is that the software ***agents*** do not have to listen in on the actual telephone conversations to detect possibly fraudulent activities.

However, from the telephone company's point of view this use of expert ***agents*** saves millions of dollars each year in lost revenues.

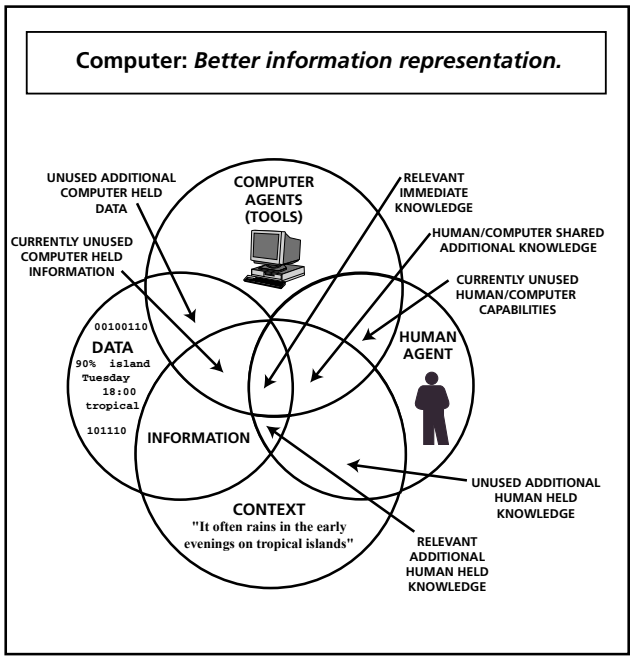

Fig.17: Evolving human-computer partnership

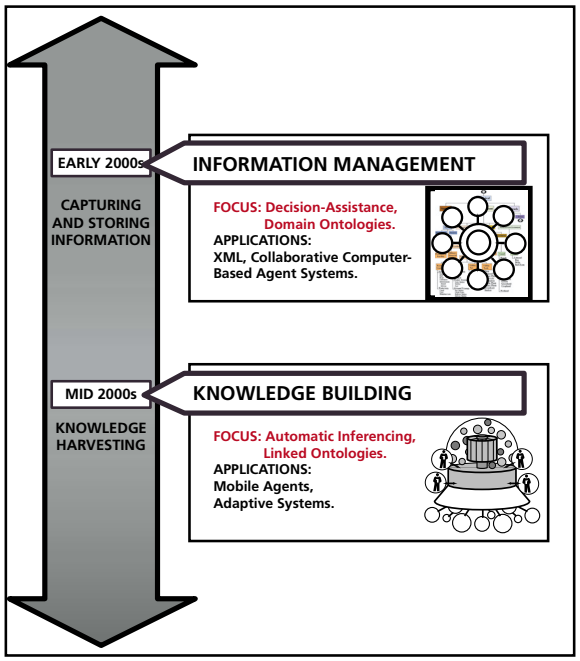

Fig.18: Evolution of business intelligence (D)

Toward the mid 2000s, we can expect some success in the linking of such ontologies to provide a virtually boundless knowledge harvesting environment for mobile agents with many kinds of capabilities. Eventually, it may be possible to achieve virtual equality between the information representation capabilities of the computer and the human user. This virtual equality is likely to be achieved not by the emulation of human cognitive capabilities, but rather, through the skillful combination of the greatly inferior artificial cognitive capabilities of the computer with its vastly superior computational, pattern-matching and storage facilities.

References

Denis S. (2000); 'Numbers'; Intelligent Enterprise, April 10 (pp. 37-44).

Ginsberg M. (1993); 'Essentials of Artificial Intelligence'; Morgan Kaufmann, San Mateo, California.

Horstmann C. and G. Cornell (1999); 'Core Java'; Vol. 1 and 2, Sun Microsystems Press, Prentice Hall, Upper Saddle River, New Jersey.

Humphries M., M. Hawkins and M. Dy (1999); 'Data Warehousing: Architecture and Implementation'; Prentice Hall, Upper Saddle River, New Jersey.

Pedersen T. and R. Bruce (1998); 'Knowledge Lean Word-Sense Disambiguitization'; Proceedings 5th National Conference on Artificial Intelligence, July, Madison WI.

Pohl J., M. Porczak, K.J. Pohl, R. Leighton, H. Assal, A. Davis, L. Vempati and A. Wood, and T. McVittie (1999); ‘IMMACCS: A Multi-Agent Decision-Support System’; Technical Report, CADRU-12-99, CAD Research Center, College of Architecture and Environmental Design, Cal Poly, San Luis Obispo, California, August.

Pohl J., A. Chapman, K. Pohl, J. Primrose and A. Wozniak (1997); 'Decision-Support Systems: Notions, Prototypes, and In-Use Applications'; Technical Report, CADRU-11-97, CAD Research Center, Design Institute, College of Architecture and Environmental Design, Cal Poly, San Luis Obispo, California, January.

Pohl J., L. Myers, A. Chapman and J. Cotton (1989); 'ICADS: Working Model Version 1'; Technical Report, CADRU-03-89, CAD Research Center, Design Institute, College of Architecture and Environmental Design, Cal Poly, San Luis Obispo, California.

Pohl K.(1996); ‘KOALA: An Object-Agent Design System’; in Pohl J. (ed.) Advances in Cooperative Environmental Design Systems, Focus Symposium: International Conference on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, Aug.14-18 (pp.81-92).

Stroustrup B. (1987); ‘The C++ Programming Language’; Addison-Wesley, Reading, Massachusetts.

Verity J. (1997); ‘Coaxing Meaning Out of Raw Data’; Business Week, June 15.

Wood A., K. Pohl, J. Crawford, M. Lai, J. Fanshier, K. Cudworth, T. Tate, H. Assal, S. Pan and J. Pohl (2000); ‘SEAWAY: A Multi-Agent Decision-Support System for Naval Expeditionary Logistic Operations’; Technical Report, CDM-13-00, CDM Technologies, Inc., San Luis Obispo, California, December.