CDM TECHNICAL REPORT: CDM-16-04

The ICDM Development Toolkit: Purpose and Overview

Jens Pohl
Kym Jason Pohl
Russell Leighton
Michael Zang
Steven Gollery
Mark Porczak

CDM Technologies Inc.
2975 McMillan Avenue (Suite 272), San Luis Obispo, California 93401

Abstract

This report provides an overview description of the Integrated Cooperative Decision-Making (ICDM) software toolkit for the development of intelligent decision-support applications. More technical descriptions of ICDM are contained in a companion CDM Technical Report (CDM-18-04) entitled: 'The ICDM Development Toolkit: Technical Description'.

ICDM is an application development framework and toolkit for decision-support systems incorporating software agents that collaborate with each other and human users to monitor changes (i.e., events) in the state of problem situations, generate and evaluate alternative plans, and alert human users to immediate and developing resource shortages, failures, threats, and similar adverse conditions. A core component of any ICDM-based application is a virtual representation of the real world problem (i.e., decision-making) domain. This virtual representation takes the form of an internal information model, commonly referred to as an ontology. By providing context (i.e., data plus relationships) the ontology is able to support the automated reasoning capabilities of rule-based software agents.

Principal objectives that are realized to varying degrees by the ICDM Toolkit include: support of an ontology-based, information-centric system environment that limits internal communications to changes in information; ability to automatically 'push' changes in information to clients, based on individual subscription profiles that are changeable during execution; ability of clients to assign priorities to their subscription profiles; ability of clients to generate information queries in addition to their standing subscription-based requests; automatic management of object relationships (i.e., associations) during the creation, deletion and editing of objects; support for the management of internal communication transmissions through load balancing, self-diagnosis, self-association and self-healing capabilities; and, the ability to interface with external data sources through translators and ontological facades.

Most importantly, the ICDM Toolkit is designed to support the machine generation of significant portions of both the server and client side code of an application. This is largely accomplished with scripts that automatically build an application engine by integrating Toolkit components with the ontological properties derived from the internal information model. In this respect, an ICDM-based application consists of loosely coupled, generic services (e.g., subscription, query, persistence, agent engine), which in combination with the internal domain-specific information model are capable of satisfying the functional requirements of the application field.

Particular ICDM design notions and features that have been incorporated in response to the increasing need for achieving interoperability among heterogeneous systems include: support for overarching ontologies in combination with more specialized, domain-specific, lower level facades; compliance with Defense Information Infrastructure (DII) Common Operating Environment (COE) segmentation principles, and their recent transition to the more challenging information-centric objectives of the Global Information Grid (GIG) Enterprise Services (GES) environment; seamless transition from one functional domain to another; operational integration to allow planning, rehearsal, execution, gaming, and modeling functions to be supported within the same application; and, system diagnosis with the objective of ensuring graceful degradation through self-monitoring, self-diagnosis, and failure alert capabilities.

An ICDM-based software development process offers at least four distinct advantages over current data-centric software development practices. First, it provides a convenient structured transition to information-centric software applications and systems in which computer-based agents with reasoning capabilities assist human users to accelerate the tempo and increase the accuracy of decision-making activities. Second, ICDM allows software developers to automatically generate a significant portion of the code, leaving essentially only the domain-specific user-interface functions and individual agents to be designed and coded manually. Third, ICDM disciplines the software development process by shifting the focus from implementation to design, and by structuring the process into clearly defined stages. Each of these stages produces a set of verifiable artifacts, including a well defined and comprehensive documentation trail. Finally, ICDM provides a development platform for achieving interoperability by formalizing a common language and compatible representation across multiple applications.

The ICDM Development Toolkit: Purpose and Overview

Table of Contents

1. Background: The Information-Centric Transformation

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.

1.1 The Quest for '*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).

1.2 The Meaning 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.

1.3 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 two decades, 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.

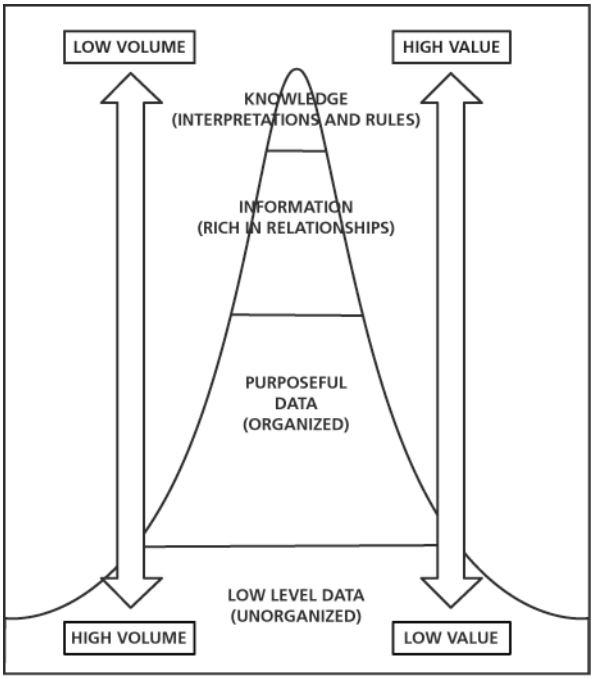


Fig.1: The ***information overload*** myth

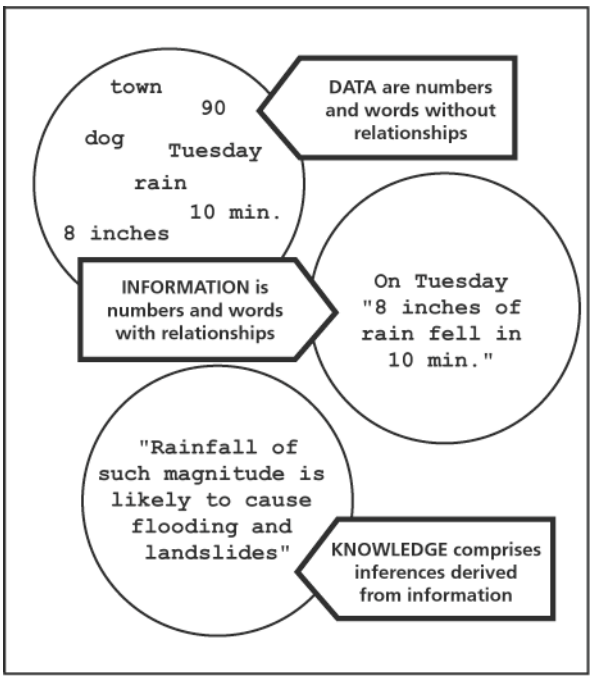


Fig.2: Data, information and knowledge

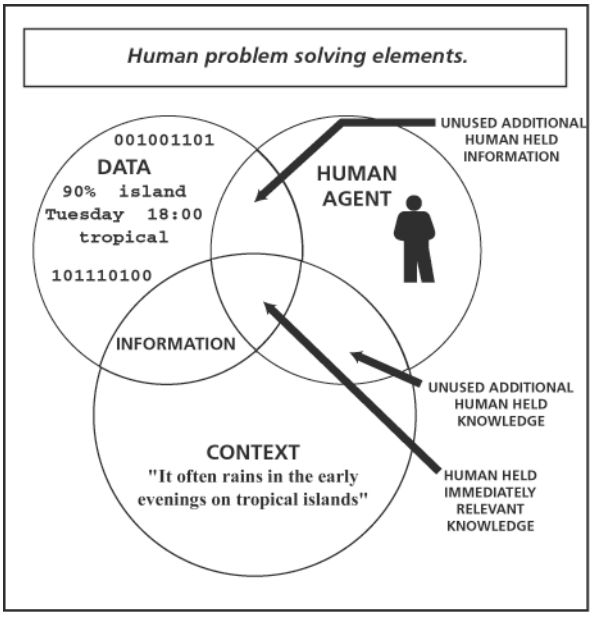


Fig.3: Unassisted problem solving

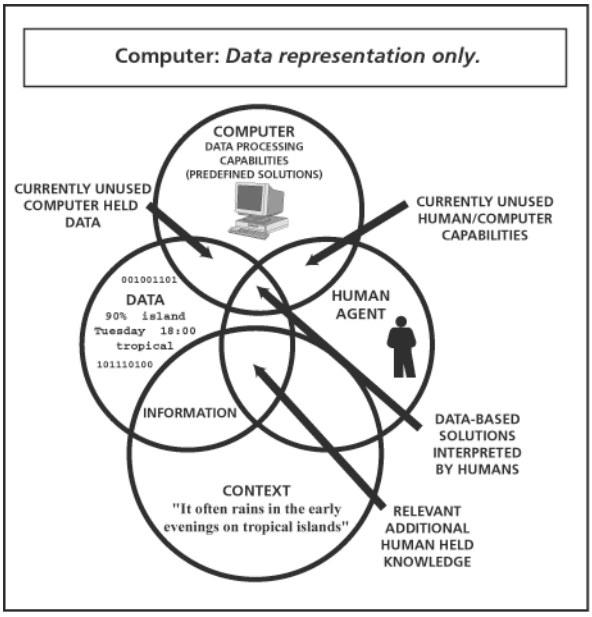


Fig.4: Limited data-processing assistance

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.

1.4 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.

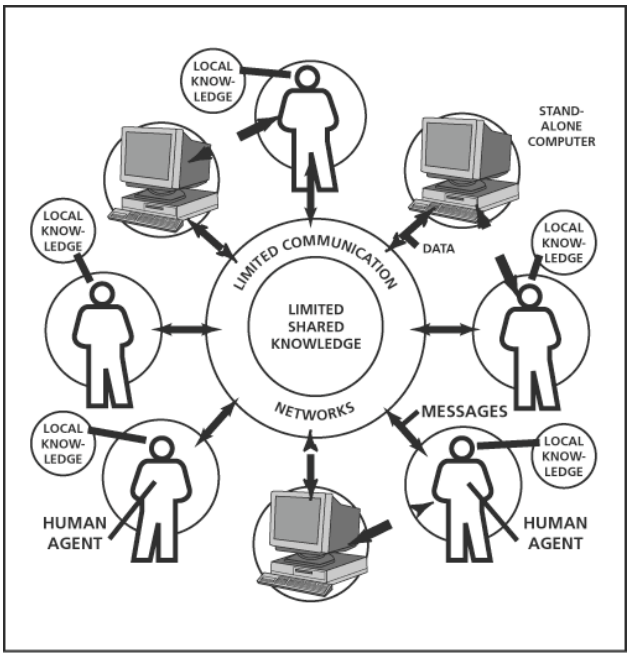


Fig.5: Limited computer assistance

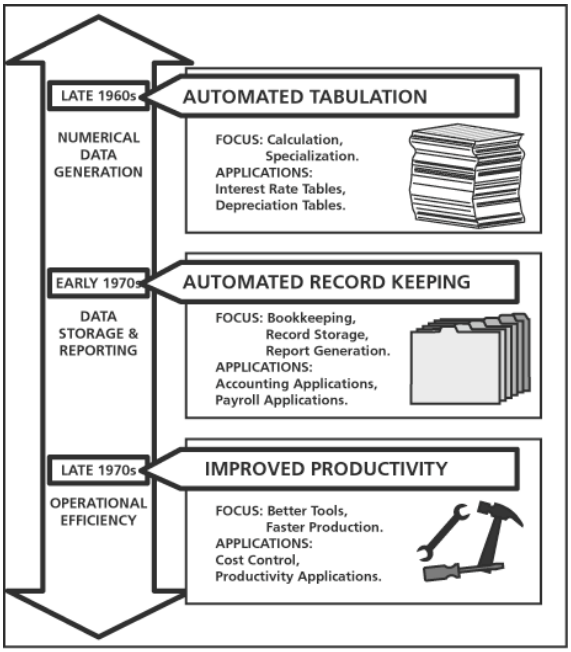


Fig.6: Evolution of business intelligence (A)

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.

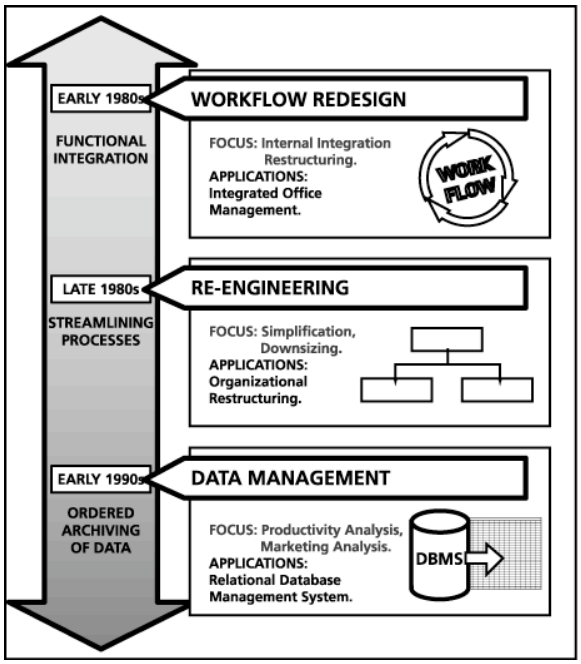


Fig.7: Evolution of business intelligence (B)

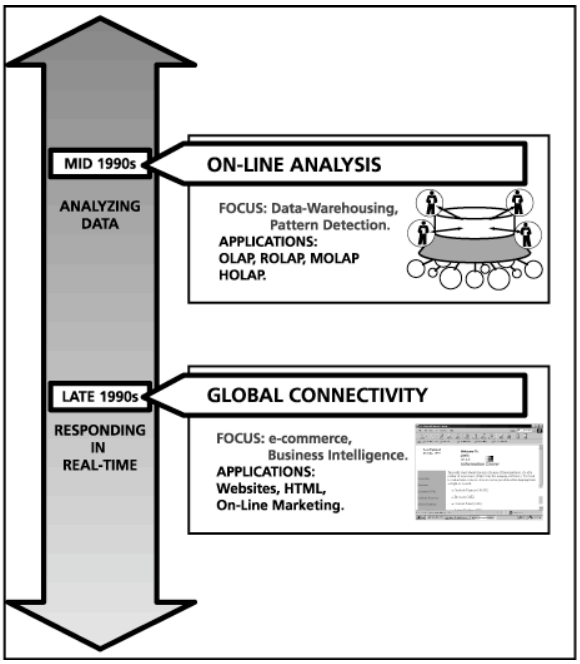


Fig.8: Evolution of business intelligence (C)

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.

1.5 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 types of objects (i.e., elements) that play a role in the domain and the allowed 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 taxonomic relationships. 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.

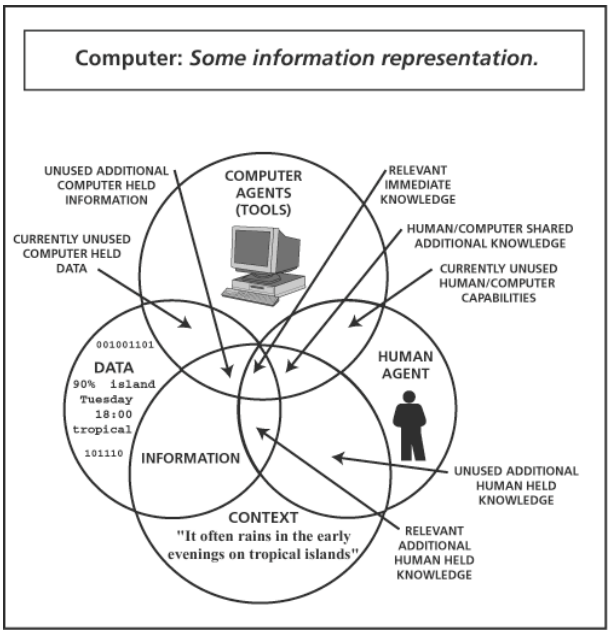


Fig.9: Early human-computer partnership

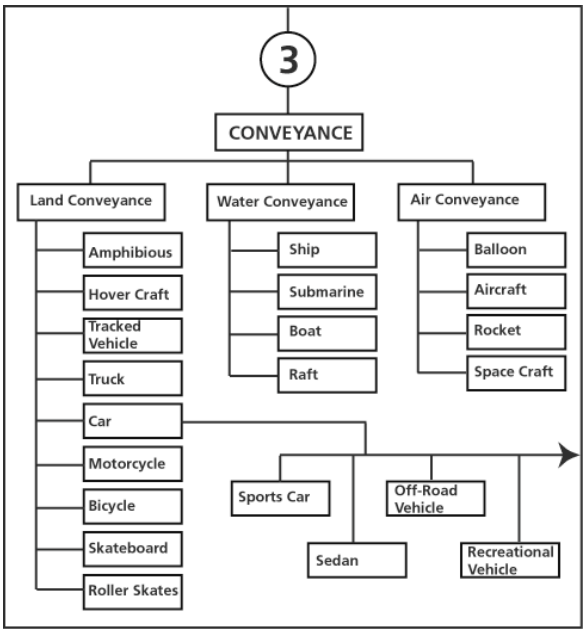


Fig.10: Branch of a typical object model

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.

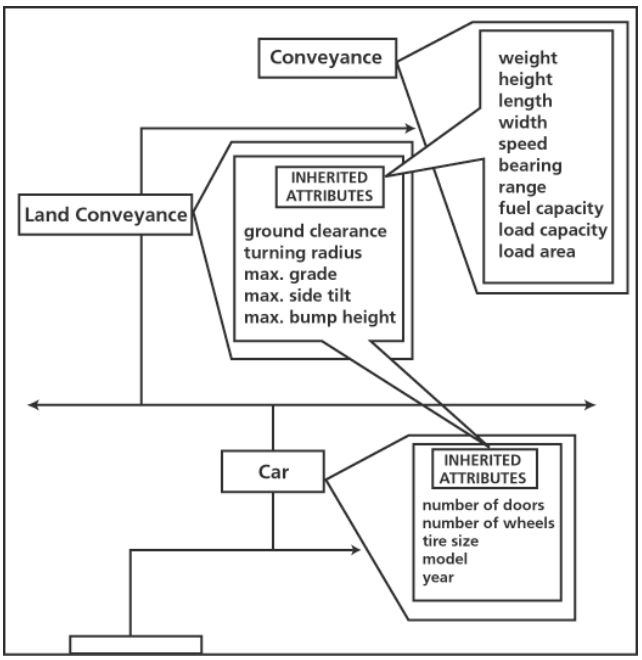


Fig.11: Object model - ***inheritance***

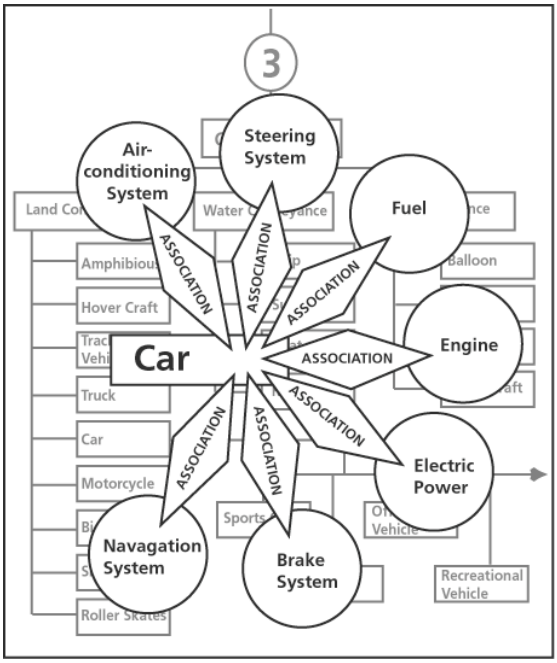


Fig.12: Object model - ***associations***

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 (i.e., taxonomic classifications). 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.

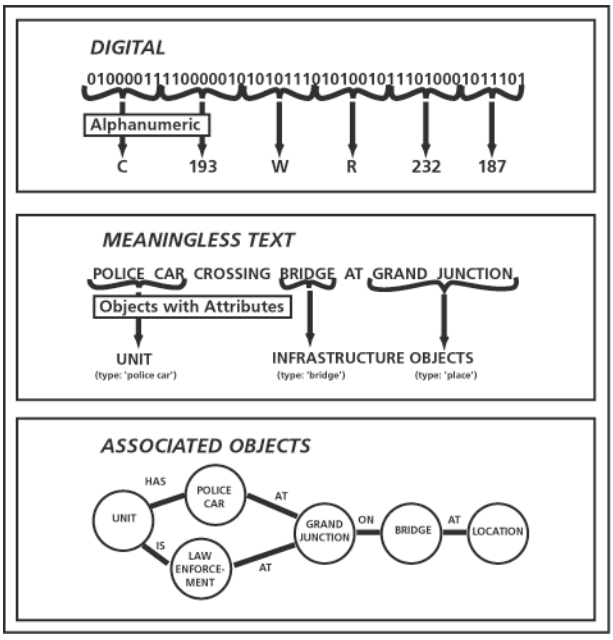


Fig.13: From ***digital*** to ***information***

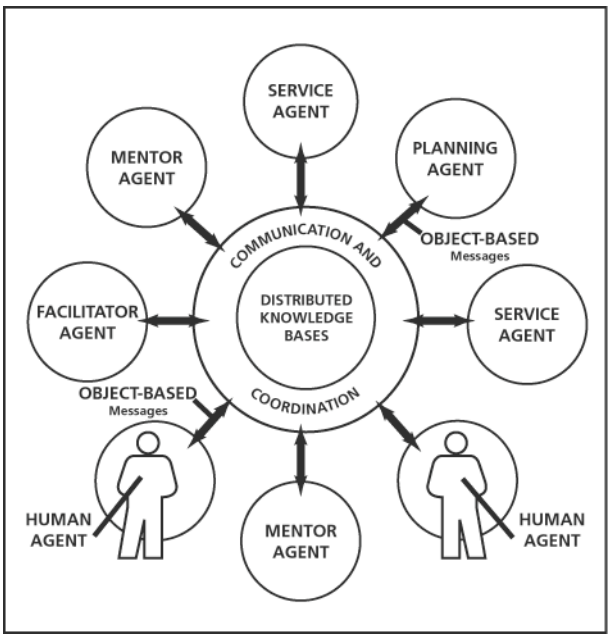


Fig.14: Types of agents

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 characteristics of its parent object, ***land conveyance***, which in turn inherits all of the characteristics 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.

1.6 The 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 (i.e., rather than hard-coded in underlying functions). 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 specific 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.

1.7 An Information-Centric 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.

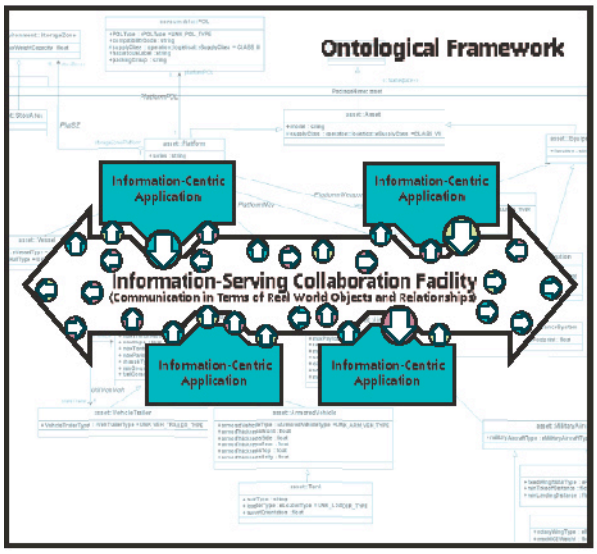


Fig.15: Information-centric interoperability.

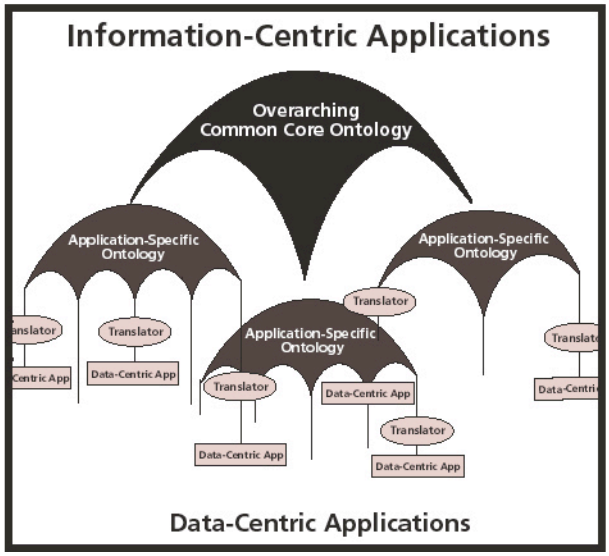


Fig.16: Transitioning to an information-centric architecture.

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.

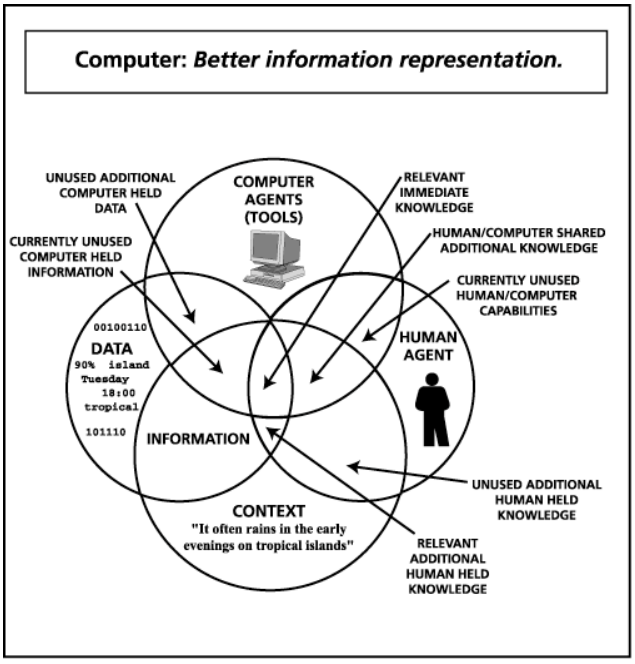
As shown in Fig.16, data-centric applications can communicate with information-centric systems through translators. The function of these translators is to map those portions of the low level data representation of the external application that are important to the decision-making context, to the ontology 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.

1.8 Promises of an Information-Centric Software Environment

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, links are established dynamically between objects: ‘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.



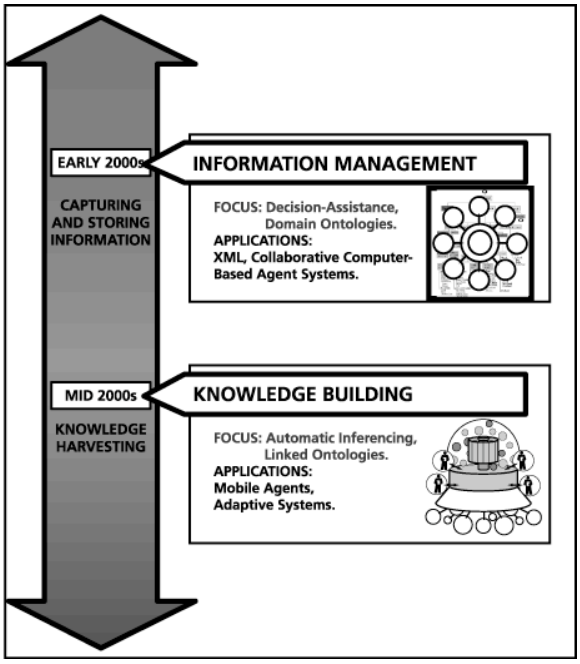


Fig.17: Evolving human-computer partnership Fig.18: Evolution of business intelligence (D)

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.

2. The ICDM Framework and Toolkit

The Integrated Cooperative Decision Making (ICDM) toolkit provides a software development framework that facilitates the design and production of distributed, multi-agent, decision-support applications. Over the past several years, the evolving ICDM framework has been used with great success by CDM Technologies, Inc. and others for the design and development of large-scale applications for military customers. Examples include the Integrated Marine Multi-Agent Command and Control System (IMMACCS) for the US Marine Corps, the logistical ICODES and SEAWAY systems for the US Army and US Navy, respectively, as well as the Shipboard Integration of Logistic Systems (SILS) and the Collaborative Advisor for Equipment Maintenance (COACH) application for the US Navy.

The core component of an ICDM-based application is a virtual (i.e., ontological) representation of the real world entities and relationships that define the ***context*** of the application domain. The representation of ***information*** (i.e., data and relationships) allows the construction of software modules, referred to as agents, capable of performing tasks that require ***reasoning*** abilities. Examples of such tasks include: the monitoring of events in dynamically changing situations; the detection of conflicts; the triggering of warnings and alerts; the formulation and evaluation of alternative courses of action; and, the collaborative assessment of situations. Typically, in ICDM-based applications the agents ***collaborate*** with each other and the human users in their monitoring, planning, and evaluation activities.

2.1 Architectural Features of ICDM

ICDM is based on a three-tier architecture that clearly distinguishes among information, logic, and presentation, components: the Information Server (i.e., ***information tier***); the Agent Engine (i.e., ***logic tier***); and, the Client User Interface (i.e., ***presentation tier***). Each of these components functions in an integrated fashion to form a comprehensive agent-based decision-support execution framework. From the viewpoint of the application environment, this framework allows multiple human decision-makers to solve complex problems in a collaborative fashion while obtaining decision-support assistance from a collection of heterogeneous on-line agents.

The core of the ICDM software development framework is the ***Information Server***, a library of objectified information that clients utilize to share information. Although clients are able to send queries to ***pull*** information, most of the information sharing occurs through a subscription service that automatically ***pushes*** any changes in information to all clients that currently include that information in their subscription profiles. Physically, the Information Server exists as a distributed object server based on the Common Object Request Broker Architecture (CORBA).

The subscription mechanism obviates the need for clients to perform multiple queries on an iterative basis to detect changes in information. Each unsatisfied query would potentially decrease resources (i.e., computing cycles) available to other application components and would therefore be wasteful of computing resources. With a more conservative approach, in which the repeated query is made on a less frequent basis, the client risks being out of date with the current state of the system until the next iteration is performed. Accordingly, the ability to ***push*** information to interested clients on a ***subscription*** basis is a highly desirable feature for providing decision-support applications with an efficient and responsive operational environment.

The ***Agent Engine*** implements the logic-tier of the three-tier ICDM architecture. Existing as a client to the Information Server the Agent Engine is capable of both receiving and injecting information. Architecturally, the Agent Engine consists of an agent server capable of serving collections of agents. These collections, or ***agent sessions***, exist as self-contained, self-managing agent communities capable of interacting with the Information Server to both acquire and inject information. For the most part, the exact nature of the agents and the particular collaborative model employed is left to the application developer. However, regardless of the types of agents contained in an agent session, agent activity is triggered by changes in the objectified application information. These objects may take the form of shared objects managed by the Information Server or local objects utilized in agent collaborations that are managed by the agent session itself. Regardless of whether agents are interacting with the Information Server or each other locally, interaction takes place in terms of objects, their attributes and relationships. This illustrates the high degree to which ICDM preserves its underlying object representation as information, allowing knowledge to be shared throughout the application environment.

The Agent Engine also supports the notion of a ***view***. A ***view*** can be thought of as representing a single investigation into solving a problem. In some cases, a ***view*** may be a conceptual perspective of reality. For example, a ***view*** may describe events and information relating to what is actually occurring in reality. In either case, ICDM maintains a one-to-one correspondence between a conceptual ***view*** and an Agent Session. This means that independent of exactly which version of reality a ***view*** represents, there exists a dedicated Agent Session providing users of that ***view*** with agent-based analysis and decision-support. Each agent of a particular Agent Session deals only with the ***view*** associated with its Agent Session. Organizing information analysis in this manner allows for an efficient and effective means of distinguishing activities relating to one ***view*** from activities pertaining to another ***view***. Unless prompted by user intervention, each set of information is completely separate from the other.

Representing the third tier of the ICDM architecture, the ***Client User Interface*** exists as a browser application (can be Web-based) that can operate in a lightweight computing environment. The Client User Interface essentially provides human users with a means of viewing and manipulating the information and analysis provided by the other two tiers. As clients of the Information Server, users have the ability to interact with each other. By either injecting or receiving information from the Information Server, users working on the same ***view*** are able to exchange design information in a collaborative manner. This type of information exchange occurs regardless of whether the relevant ***view*** represents the main effort or exists as a localized solution attempt explored by a subset of users. All information and analysis remains localized within its particular ***view*** unless explicitly copied into another ***view*** as a user-initiated action. In this manner, no informational or analytical collisions occur between conceptual ***views*** without user-based supervision and subsequent reconciliation.

2.2 The ICDM Software Development Process

The ICDM framework of development tools is a direct outcome of almost 20 years of experience in the design and development of agent-based applications and systems. It has evolved from a powerful but loosely defined set of tools into an integrated tool-based software development environment in which significant server and client-side code components can be generated automatically.

The ICDM software development process commences with the design of an outline ontology that

is progressively extended during the life cycle of the application, to support the full scope of the functional context of the application domain. Utilizing any one of several commercially available UML (Uniform Modeling Language) modeling tools the development of the ontology involves the selection of objects, the establishment of object relationships, the exploration of functional notions and their translation into ontological components, and the identification of the principal object attributes.

The development of the initial ontological framework precedes the design of any detailed application components and certainly any code development. It must be supported by a comprehensive and well-documented knowledge acquisition process that defines user needs within the functional application domain. Once the initial ontology is available it serves as the application context in the automatic generation of the principal server and client-side components of an ***application engine***. Up to this point in the development process no manual coding tasks have been undertaken. The application engine represents an integrated, executable application environment incorporating all principal internal services such as semantic network manager, subscription manager, inference engine, agent manager, and so on.

The remaining missing components that must be coded manually include the individual domain-specific agents and user-interface features, the customized mapping services of the translators that allow external systems to share data with the information-centric application, and any required capabilities that may not have generic counterparts in the ICDM tool-set.

2.3 Origin, Proprietorship, and Licensing

The initial formulation and development of ICDM was performed by individuals of the Collaborative Agent Design Research Center (CADRC) at the California Polytechnic State University, San Luis Obispo (Cal Poly), who then formed CDM Technologies, Inc. (CDM) as a supportive commercial entity. In January 2000 ownership of the ICDM Toolkit was transferred from the original individual owners to CDM.

CDM and the CADRC at Cal Poly have, since the formation of CDM in 1994 and will continue in the future, to work closely together on most ICDM-based projects for the US Government and, in particular, the US Department of Defense (DoD). This university-approved unique relationship between the CADRC and CDM promotes the transition of advanced technologies to government and industry users. It also insures the continued availability of ICDM independently of the continued existence of CDM. The CADRC was granted by CDM in January 2000 an unrestricted, free license to use, modify, and maintain ICDM. Accordingly, in the event that at some future date CDM should cease to exist, licensees will continue to have assured access to ICDM expertise, development capabilities, and support through the CADRC as an official Cal Poly University Research Center.

The ICDM Toolkit can be licensed directly from CDM Technologies, Inc. together with variable software maintenance contract terms including technical support and consultation services. The ICDM Toolkit consists of the following principal components and a larger number of secondary accessory tools and reusable components:

- ICDM object-based schema implementation.
- Library of ICDM code and report generating tools.
- ICDM information server.
- ICDM persistence service.

- ICDM agent engine.
- ICDM client user-interface components.

3. The Underlying Design Principles

For the past 16 years the Collaborative Agent Design Research Center (CADRC) at the California Polytechnic State University (Cal Poly) and more recently CDM Technologies, San Luis Obispo have pursued the design and development of agent-based decision-support systems. Throughout this journey the CADRC and CDM have relied on a suite of development tools that greatly assist in the creation and management of such systems. This suite of tools is known as the Integrated Cooperative Decision-Making (ICDM) software development toolkit (Fig.19).

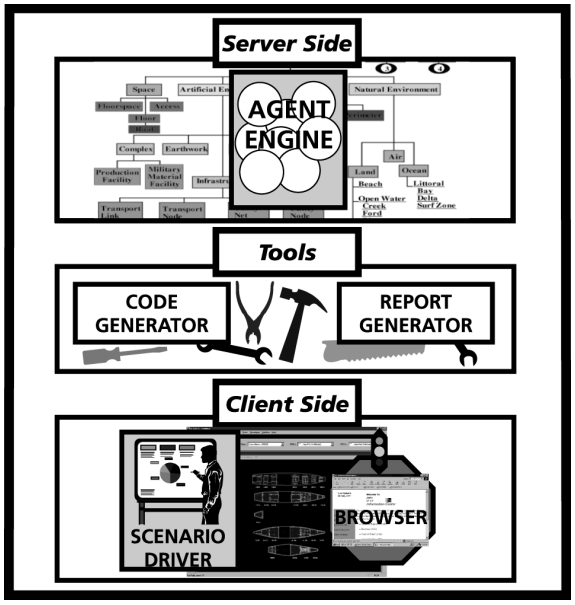


Fig.19: ICDM Development Toolkit

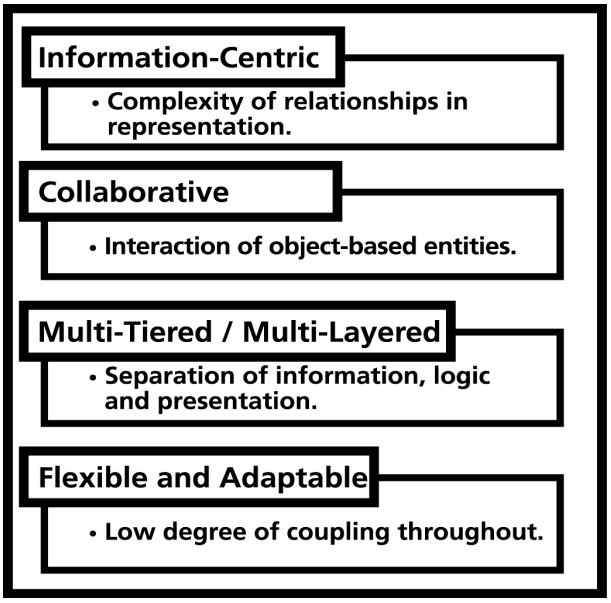


Fig.20: ICDM Design Principle

Not only does ICDM function as an accelerator (i.e., rapid development) and stabilizer (i.e., built-in robustness and fault tolerance) in the development of decision-support systems, but it also provides a concrete vehicle for representing the key concepts and philosophies that the CADRC and CDM have found to be useful for the success of these types of systems (Pohl et. al., 2000; Pohl, 1997). This Section focuses on the key design principles on which ICDM is founded; namely, collaboration-intensive, context-based representation, flexibility and adaptability, and multi-tiered, multi-layered architecture (Fig.20).

3.1 Collaboration-Intensive

Certainly in the real world, collaboration among decision-makers and experts is a critical ingredient in making educated and effective decisions. This is especially true when operating across an extensive and varied set of domains. Through years of research in collaborative design the CADRC has found that this same quality extends to the realm of agent-based decision-support systems. Conceptually, the systems developed by the CADRC and CDM consist of dynamic collections of collaborators (both human and software-based) each playing a role in the collective analysis of a problem or situation and the consequential decision-making assistance required in formulating an accurate assessment and /or solution.

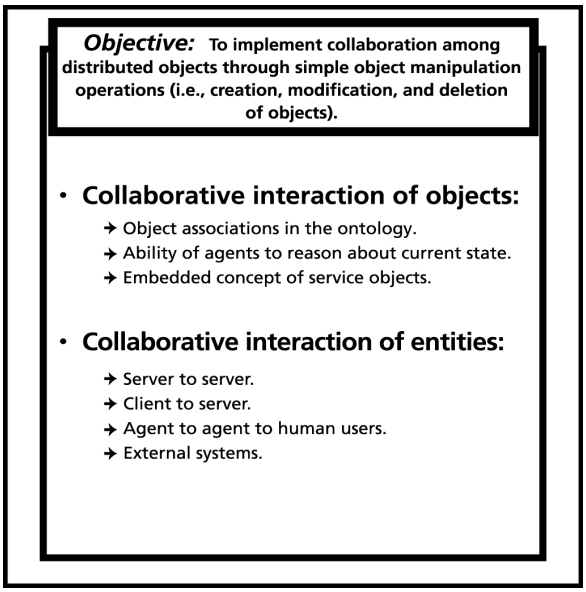


Fig.21: Design Principle: *Collaboration*

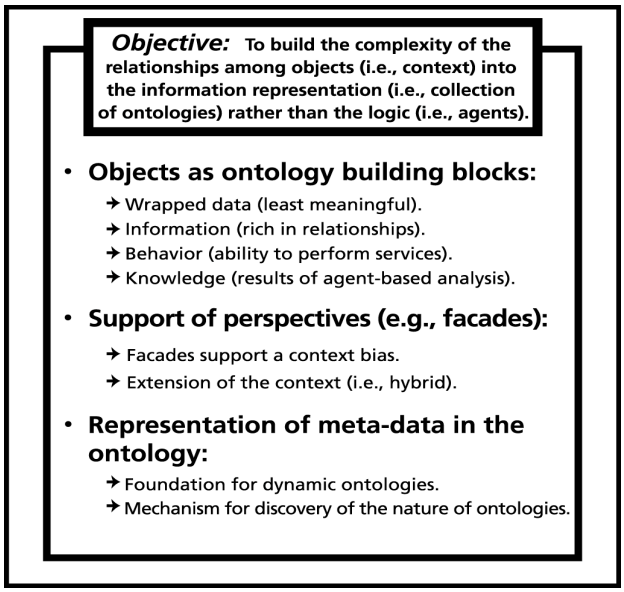


Fig.22: Design Principle: *Information-`Centric (Context)*

Whether human or software-based, collaboration within an ICDM-based system occurs in terms of a descriptive ontology (Chandrasekaran et. al., 1999). Until recently these ontologies were limited to describing information and knowledge that represents various aspects of the domain(s) over which the system is to operate. For example, in the domain of architectural design the applicable ontology would describe such notions as spaces, walls, accessibility, appropriate lighting, and so on. Although effective and certainly a fundamental element of an information-centric system a considerable portion of the system still remains in a form not necessarily supportive of highly collaborative environments. Further, these non-ontology based components require separate and dedicated interfaces along with specialized management. A number of the services that collaborators within an ICDM-based system interact with (e.g., time, query, subscription, execution, reasoning, etc.) were still presented as client-side adjunct-based interfaces requiring additional management to support collaboration. For example, if two clients wish to share or discuss the same subscription profile, a separate mechanism for identifying and referencing the collection of interests is required. In this case, the interface would be the client-side Application Programming Interface (API) maintained by the subscription service itself. Although certainly possible, supporting such specialized functionality requires the particular services (i.e., the subscription service in this case) to present and manage a specific API to expose or match global references. Although subtle in nature, complexity such as this can easily escalate when considering the high degree of collaboration inherent in multi-agent decision-support systems.

Recently, however, CDM has overcome this limitation by taking the notion of *objectified collaboration* to the next level (Fig.21). This approach extends the once solely information-based ontology to include behavioral aspects of the decision-support system. More specifically these constrained behavioral objects constitute the services within the decision-support system (i.e., the services themselves are represented in the collaborative ontology in the same manner as

information and knowledge). The only difference is that these distributed and shareable objects offer behavior in addition to information. As a result collaborators are able to interact with these services through the same distributed object operations that they would perform on the information and knowledge objects. Any constraints identified in the behavior are enforced by the standard ontology management facility. The operations that can be performed on these ontology-based objects consist of the basic creation, deletion, and modification functionality. To support the aforementioned example in which two collaborators wish to reference and discuss aspects of the same subscription profile, the two collaborators would treat the profile in question as just another set of multi-faceted, shareable distributed object. In other words, similar to the manner in which rich information models, or ontologies are used as a basis for collaboration this notion is extended to include interaction and collaboration across the services that constitute the system itself. The effect is essentially that interaction with and collaboration across information, and now behavior (e.g., services), is reduced to a basic set of object manipulation capabilities. In this sense, an object is-an-object is-an-object. The only difference is that some distributed, shareable objects offer information and some offer behavior. The client-side portion of the ontology replaces the need for specialized client-side functionality.

3.2 Context: Information-Centric

Representation can exist at varying levels of abstraction (Fig.22). The lowest level of representation considered in this paper is *wrapped* data. Wrapped data consists of low-level data, for example a textual e-mail message that is placed inside some sort of an e-mail message object. While it could be argued that the e-mail message is thereby objectified it is clear that the only objectification resides in the shell that contains the data and not the e-mail content. The message is still in a data form offering a limited opportunity for interpretation by software components.

A higher level of representation endeavors to describe aspects of a domain as collections of inter-related, constrained objects. This level of representation is commonly referred to as an information-centric ontology. At this level of representation context can begin to be captured and represented in a manner supportive of software-based reasoning. This level of representation (i.e., context) is by far the most empowering design principle on which ICDM is based. Further, as mentioned in the previous section portions of this context may be extended to exhibit behavior. In addition to services, however, distributed behavioral objects can also be employed as a mechanism for supporting the notion of facades.

Existing as one of the fundamental design patterns employed in object-oriented design (Pohl K. 2001) facades provide a level of derivation attained from the particular representation or ontology on which they are based. In the case of ICDM and the type of ontologies it manages facades offer a method of supporting and managing an alternative *perspective* from that modeled in the ontology from which they are derived (Pohl K. 2001). In other words, ICDM-based facades allow the perspective inherent in a particular model of a domain to be augmented, or in some way altered to support a more appropriate (i.e., to the façade user) representation of the concepts, notion, and entities over which that *user* is operating (Fig.23). Note that *user* in this sense refers to any accessing component. While certainly useful in systems supporting multiple perspectives caution must be employed in preventing abuse by introducing inconsistency and unnecessary duplication.

Facades can also be utilized to support real-time calculations. In this sense, the façade derivation would involve a calculation or algorithm perhaps based on one or more attributes of the base

object(s). For example, consider an architectural space exhibiting length, width, and height described in American pound/foot units which is to be accessed by a design system that only understands Metric kilogram/meter units and also requires space volumes. Utilizing ontology-based facades a model could be developed in which, not only the length, width, and height, but also the volume of the space could be calculated and presented to the design system in terms of Metric units. Although there are a number of approaches to supporting calculated attributes in the case where an alternative perspective is to be supported, the façade approach permits an extensible (i.e., one perspective extended from another) and encapsulated (i.e., easily maintainable) solution.

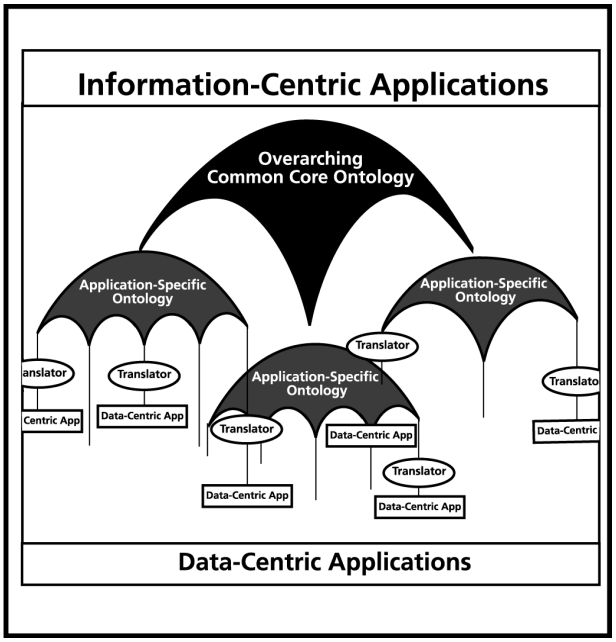


Fig.23: Ontology Cluster
Supporting Multiple Levels of Abstraction

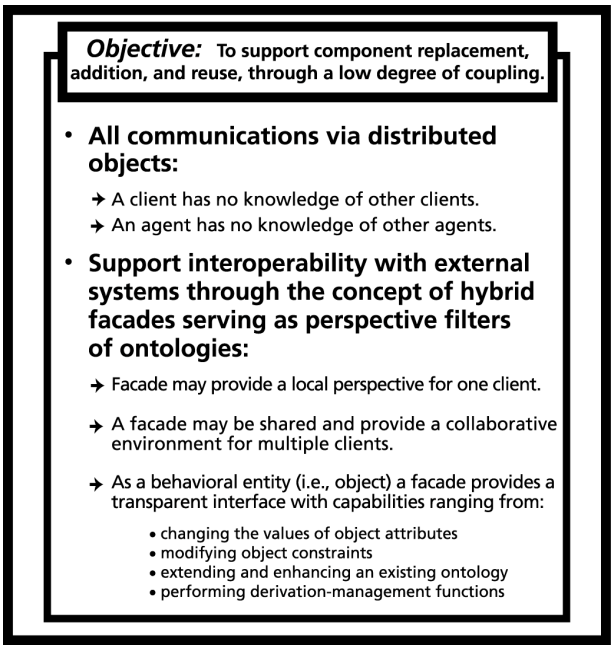


Fig.24: Design Principle:
Extensibility and Adaptability

3.3 Extensibility and Adaptability

One of ICDM's primary goals is to support a high degree of flexibility in respect to the configuration of its components both at the development and execution levels. ICDM supports the addition, replacement, and reuse of software components in the context of agent-based, decision-support systems, and achieves this goal by reducing inter-component coupling to an absolute minimum (Fig.24). There are two key ICDM properties that permit this flexibility. First, all collaboration between clients takes place via, and in terms of the informational ontology (i.e., distributed objects). No direct communication exists between collaborators. The result is a collaborative environment in which client identities are essentially irrelevant in respect to this process. This low degree of coupling permits the reconfiguration (i.e., component addition, removal, or replacement) of collaborating components at any point during an execution session.

The second property deals with the manner in which clients access and interact with the ontology. ICDM offers a standard interface component known as the Object Management Layer (OML) which both shields accessors from the complexity of ontology management as well as provides an abstracted view of the ontology. Clients of OML interact with the ontology via

object wrappers (POW) based on a set of corresponding ontology-specific templates. Promoting the notion of adaptability, these templates are discovered by OML as a runtime activity. The resulting support for dynamic definition permits elements of the ontology to be extended, eliminated, or even redefined during the course of a runtime session.

Apart from the ability to adapt to an evolving definition of a domain, adaptability is also supported in interaction with external systems. This level of adaptability functions in conjunction with the concept of façades mentioned earlier. Replacing the classical approach of building a dedicated and separate translation bridge between collaborating systems, ICDM promotes the incorporation of such translation into the ontology itself. In other words, using ICDM's support for ontology-based facades, translation or derivation of each system's perspective can be encapsulated and managed solely within façade objects. The resulting translation facility exists as a set of behavioral façade objects accessed and manipulated in a manner no different than is applied to other ontology objects. The result is an elegant design where support for translation-based communication between disparate systems is seamlessly incorporated as part of the ontology.

3.4 Multi-Tiered and Multi-Layered

The forth design principle to which ICDM adheres addresses the architectural organization of ICDM-based systems. More specifically, this principle identifies distinct separations between areas of functionality at both the conceptual (i.e., tier) level and the more concrete (i.e., layer) level. Conceptually, the architecture of an ICDM-based decision-support system is divided into three distinct tiers namely, information, logic, and presentation. To manage its particular domain each tier contains a number of logical layers that work in sequence (Fig.25).

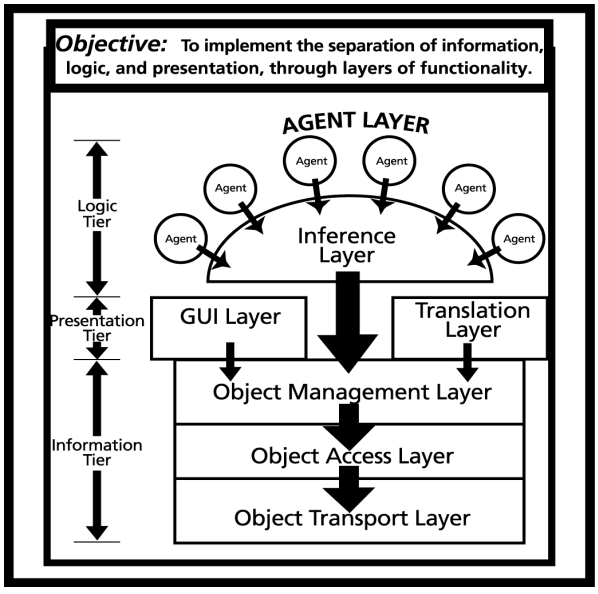


Fig.25: Design Principle:
Multi-Tiered, Multi-Layered Architecture

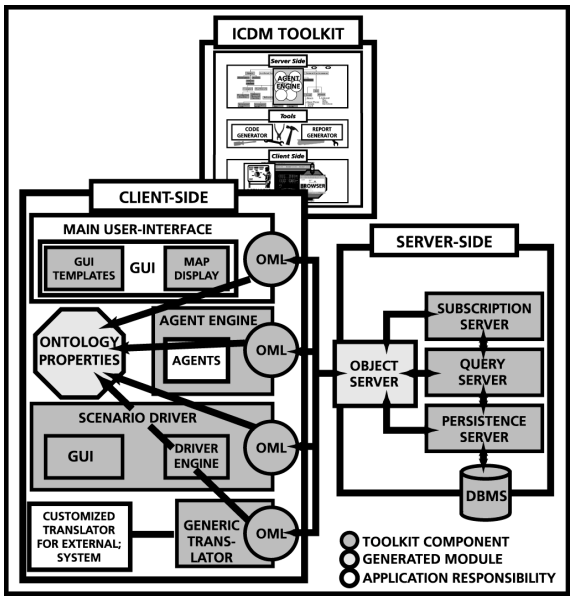


Fig.26: ICDM-Based System
Component Types

As the name suggests the information tier houses both the information and knowledge (i.e., ontology) being operated on in addition to all of the mechanisms needed to support management, transport, and access. The information is further delineated into layers. The first of these is the Object Management Layer (OML) described in an earlier section. Below the OML resides the Object Access Layer (OAL) responsible for managing access to the information tier. The OAL exists as a level of abstraction below OML and interfaces directly with the Object Transport Layer (OTL). Based on the CORBA specification (Mowbray and Zahavi 1995) the OTL is responsible for communicating the various requests and subsequent replies for distributed information and behavior issued through the OAL throughout the system. The OTL is the only layer that forms a dependency on an underlying communication protocol. As such, support for alternative communication facilities can be implemented with minimal impact on either the OAL or the OML. This is an excellent example of the benefits of a layered architecture in supporting component reuse and replacement.

The Logic Tier contains the business rules (i.e., agents) and analysis facilities by which these rules are managed. Although extensible to include other forms of reasoning the current version of ICDM focuses on opportunistic rule-based analysis. Regardless of which form of reasoning is employed this capability is supported by two layers namely, the Business Rule Layer (BRL) and the Business Engine Layer (BEL). The BRL is primarily system-specific and contains the agent-based analysis facilities resident in the system. Execution of agents is in turn managed by the BEL. To integrate the Logic Tier with the Information Tier the BEL interfaces with OML permitting the agents to both access and contribute to the ontology.

The final tier is the Presentation Tier. This tier is responsible for interfacing with the various users of the system. In this sense a user may be a human operator or an external system. In the case of a human operator support is provided through a Graphical User Interface Layer (GUIL) that presents and promotes interaction with the contents of the Information Tier. In the case of an external system, support takes the form of a Translation Layer (TL) that manages the mapping of representations between systems. Like the GUIL, access to and from the Information Tier is supported by OML.

3.5 The ICDM-Based Application Environment

As a toolkit for the development of agent-based, decision-support systems ICDM supports three types of components each working in conjunction with the others to form a complete decision-support system, namely: the toolkit facilities; the automatically generated modules; and, the application-specific code that must be created manually (Fig.26). The first type of component is automatically generated from the ontology. Among the generated artifacts is a property file that contains detailed characteristics of each object described in the ontology. These properties are used to configure the second type of component. Configuration of the second type of component takes place during runtime and essentially conforms this category of components to the specific system in which they are operating. The third type of component is system-specific and the responsibility of the particular project. This set of components primarily includes the agent rules and user-interface. Together, these three types of components are integrated at the project level to formulate a specific agent-based, decision-support system.

Adhering to the design principles of collaboration, context-based representation, extensibility and adaptability, and a multi-tiered, multi-layered architecture, ICDM can be effectively utilized in the rapid development of agent-based decision-support systems spanning a variety of complex

domains. ICDM has been successfully employed by the CADRC and CDM in the development of decision-support systems ranging from architectural design (e.g., ICADS and KOALA, (Pohl et. al. 1991; Pohl K. 1996)) to tactical command and control (e.g., IMMACCS and FALCON, (Pohl et. al. 1999)). Profiting from being founded on a framework embodying the principles described in this Section each such decision-support system exhibits the key qualities (e.g., collaborative, high level representation, and tools as opposed to predetermined solutions) that are vital to the implementation of effective agent-based, decision-support systems.

4. The ICDM Architecture

The initial development of the ICDM toolkit was undertaken by individuals prior to the formation of CDM Technologies, Inc., with the objective of providing a formalized architecture together with a set of development and execution tools that could be utilized to design, develop, and execute agent-based, decision-support applications. The core element of this process is the development of an ontological framework to provide a relationship-rich model of the various system and application domains in which the proposed application is required to operate. Based on a three-tier architecture, ICDM incorporates technologies, such as distributed-object servers and inference engines, to provide a framework for collaborative, agent-based decision-support systems that offer developmental efficiency and architectural extensibility.

Throughout the past decade CDM Technologies Inc., in association with the Collaborative Agent Design Research Center (CADRC) at Cal Poly (San Luis Obispo, California), has been engaged in the design and development of agent-based, decision-support systems from a decidedly pragmatic standpoint (Pohl et al. 1997). As a result of these efforts, CDM Technologies has developed a manifesto of sorts describing a collection of criteria which we consider to be fundamental to the development of agent-based, decision-support systems (Pohl 1997).

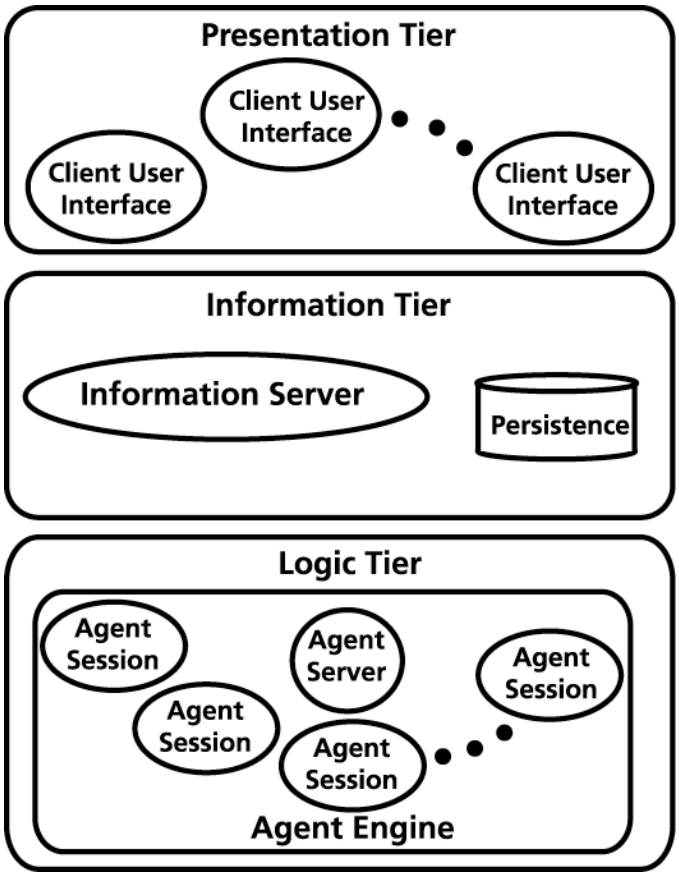


Fig.27: The three-tier architecture of the ICDM toolkit

First and foremost on this list of design criteria is the need for an object-based representation of information that will allow software modules (i.e., agents) to reason about events in the problem space. Within the context of a top-down symbolic approach (as distinct from a bottom-up

connectionist approach (Minsky 1991)) information processed within the system is described as objects with attributes, behavior, and most importantly relationships. Collectively, such symbolic descriptions form an application's information object model or ontology (Fowler and Scott 1997). This requirement not only applies to the modeling of information but, at times, is even portrayed in the representation of the agents themselves. Without such an objectified representation that allows critical informational relationships to be captured, determination of the meaning and implication of information becomes extremely difficult, if not impossible.

After several implementations it became clear that to take full advantage of such objectified representation, a supportive framework needed to be established. A framework that centers around objects. To satisfy this need the ICDM framework was developed and continues to evolve as an architecture, together with a set of development and execution tools that can be used to design, implement, and execute agent-based, decision-support applications.

As mentioned previously in Section 2.1, the ICDM model is based on a three-tier architecture that draws a distinction between information, logic, and presentation (Gray et al. 1997). These tiers are represented by the three major components comprising the ICDM model: the ***information tier*** consisting of a collection of information management servers (i.e., Information Server, Subscription Server, etc.); the ***logic tier*** represented currently by an Agent Engine; and, the ***presentation tier*** or Client User Interface (Fig.27). Each of these components functions in an integrated fashion to form a comprehensive agent-based decision-support execution framework. This framework allows multiple human decision-makers to solve complex problems in a collaborative fashion obtaining decision-support assistance from a collection of heterogeneous on-line agents.

4.1 Information Server

The core of the ***information tier*** is the Information Server. Conceptually, the Information Server represents a library of objectified information that clients utilize to both obtain and contribute information. The only difference is that clients can obtain this information in, not only a ***pull*** fashion (i.e., query service), but can also have the Information Server ***push*** them information on an interest basis (i.e., subscription service). Physically, the Information Server exists as a distributed object server based on the Common Object Request Broker Architecture (CORBA) (Mowbray and Zahavi 1995).

Distributed object servers, forming the basis of the Information Server, are designed to service client requests for information. The knowledge of exactly where the information resides and how it can be retrieved is completely encapsulated inside the object server. This means that clients need not be concerned with who has what information and in what form that information exists. This feature becomes instrumental in providing an environment where collaborative application components operate in a decoupled manner via the Information Server.

Regardless of the native representation of information, distributed object servers can be used to present information to clients in the form of objects. However, this does not discount the need for information to be modeled in its native form as high-level objects, portraying behavior and conveying relationships. While on the surface this representational morphing capability of object servers seems promising, in practice it can be quite misleading. If the information is not represented at a high level upon its conception, such objectification amounts to little more than wrapping data in communicable object shells. These shells fail to convey any more insight into the meaning or implication of the information than was present to begin with in its original form.

Although in the future there may be potential for successful research efforts in this area, at present, unless information is originally modeled as objects, knowledge-oriented applications prove to gain little from a distributed object server feature.

However, applications that do model information as high-level objects stand to gain considerably from employing distributed object servers. Under these conditions distributed object servers preserve the purely objectified representation of information as it moves throughout the system. This is due to the fact that the internal mechanisms of distributed object servers process information as objects themselves.

A powerful method that can be used to obtain information from the ***information tier*** utilizes the notion of subscription. Clients can dynamically register standing subscriptions that are again described in terms of the application's ontological system. For example, a client may request to be notified whenever the available inventory quantity of a particular supply item falls below a specified threshold value. Once registered, the Subscription Server continually monitors this condition. When satisfied, the Subscription Server essentially ***pushes*** the results to whichever client has indicated an interest (i.e., registered an appropriate subscription). The alternative to this subscription mechanism would be to have interested clients perform the same query on an iterative basis until such a condition occurs. Each unsatisfied query will potentially decrease resources (i.e., computing cycles) available to other application components. If a client takes a more conservative approach by which the repeated query is made on a less frequent basis, the client risks being out of phase with the current state of affairs until the next iteration is performed. With this in mind, the incorporation of a mechanism that pushes information to interested clients becomes a very valuable facility in providing decision-support applications with an efficient, up-to-date execution environment.

A subscription service essentially monitors a dynamic set of client interests or subscriptions. In this sense, a subscription can be thought of as a ***standing*** or continuous query. Similar to individual queries, interests can be described in terms of information values and constrained events. Once posted, these interests are monitored by the subscription service on a continuous basis. If satisfied the subscription service notifies all interested parties of the situation. This notification can even be coupled with the ***pushing*** of contextual event information to appropriate subscribers. However, in many cases the relevant information a subscriber seeks is that the event has occurred and the subscriber may not in fact be concerned with actual event context. In this case it is more efficient to decouple the notification of interest satisfaction and the conveyance of contextual information. If desired, such information can be obtained through a series of follow-on queries issued by the subscriber.

Combination of an object-based representation (i.e., ontology) with an inference engine provides a very powerful subscription capability within the necessary object-serving communication facility. We have had some experience with the development of subscription services utilizing the CLIPS rule-based inference engine developed by NASA (NASA 1992). However, many other inference engines are commercially available. CLIPS is offered, essentially free of charge (i.e., cost of documentation only) to US Government agencies and their contractors, in source code form.

In all rule-based inference engines predicate logic is described in terms of rules, potentially complex patterns together with related actions. Each rule has a pattern describing a set of conditions and a subsequent action to execute upon satisfaction of the conditions. To efficiently manage the matching of potentially high volumes of patterns across equally high and ever-changing pools of information, CLIPS employs an extremely efficient scheme known as the Rete

algorithm (Forgy 1982). As a result the varying degree of pattern satisfaction across large sets of rules and dynamic sets of information can be maintained in the context of a near real-time decision-support system. It is this efficient and extensive pattern matching ability that makes such a mechanism an excellent candidate for forming the core of a robust, ontology-based subscription service.

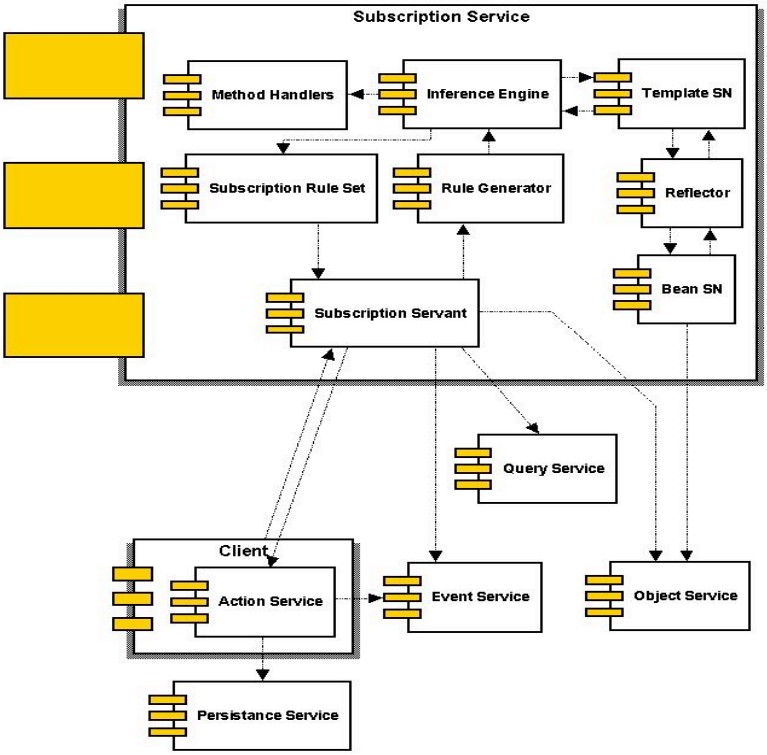


Fig.28: Inference engine-based subscription service architecture

The basic components comprising a subscription service architecture are shown in Fig.28. Each component works in conjunction with the others to effectively manage the dynamic set of subscriptions. If implemented within a CORBA-based environment this architecture adheres closely to the application server design pattern (Ayers et al. 1999). In such a pattern both information and functionality are served to a dynamic set of clients as sharable, collaborative objects. Following this pattern, the subscription service is presented to clients in the form of subscription objects that can be instantiated. These subscription objects embody the same set of qualities as any CORBA object, and can be represented in the ontology as a subscription/notification domain (Fig.29).

To invoke the subscription service a client may either instantiate a subscription object or instead chose to utilize an existing subscription registered by another client (i.e., subscription objects enjoy the CORBA quality of being sharable). Regardless of method, during this process the subscriber also associates a local action object to the subscription (Fig.30(a)). It is this action that the client wishes to have executed upon subscription satisfaction. Each time a new subscription is created the Rule Generator component of the subscription service constructs a corresponding

CLIPS rule. The content of this rule represents the characteristics of the particular subscription. This rule is then placed under the management of the CLIPS inference engine that in turn monitors the pattern for satisfaction. If a match occurs the action portion of the rule (not to be confused with the client action object) triggers the associated client action.

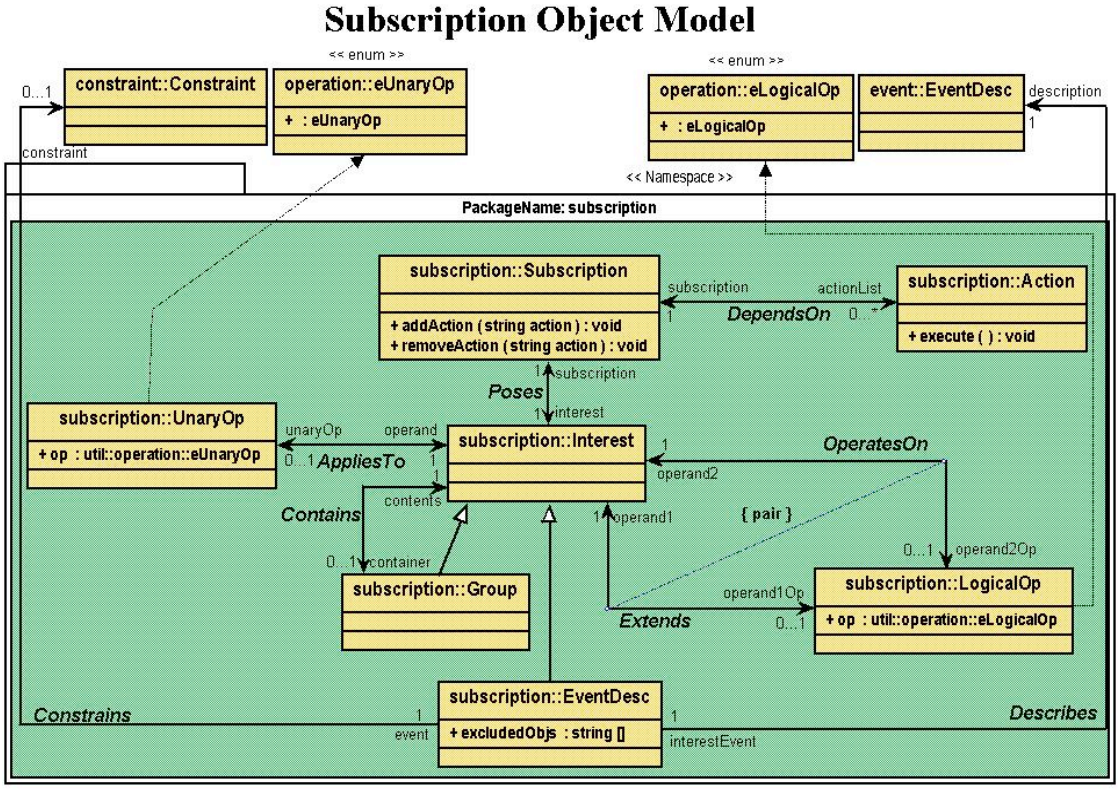


Fig.29: Subscription service domain ontology described

In the subscription service architecture described here, client action objects exist as specialized CORBA objects. However, while subscription objects are implemented within the scope of the subscription service, action objects are implemented within the context of the subscribing client. This introduces a powerful capability inherent in CORBA-based architectures.

In a CORBA-based paradigm the distinction between client and server is somewhat blurred. Each application component has the potential of being a client in one sense and a server in another. The notification mechanism outlined in this design makes significant use of this feature to permit asynchronous communication between the subscription service and its clientele. Once the subscription service identifies that an interest has indeed been satisfied the subscription service then becomes a client to the action server part of the subscriber it intends to notify. In a CORBA-like fashion, the subscription service remotely executes the ***notify*** method of the action object of each relevant client (Fig.30(b)). Once they are executing within the context of their action objects, notified subscribers may determine the most appropriate course of action to react to the event.

By employing an inference engine as the core of the subscription service two significant capabilities are achieved. First, subscribers in a decision-support application can now exploit the powerful and extensive pattern matching functionality inherent in a rule-based inference engine.

The degree and complexity of subscriber interests are constrained only by the extent of the ontological system on which they operate. Second, a strong similarity can be drawn between subscriptions comprised of extensive interests together with specific actions and the powerful pattern/action architecture of an expert system rule. In this manner, the subscription service allows decision-support applications to be described as collections of ***distributed*** expert system rules with the inherent benefits of autonomous and opportunistic capabilities.

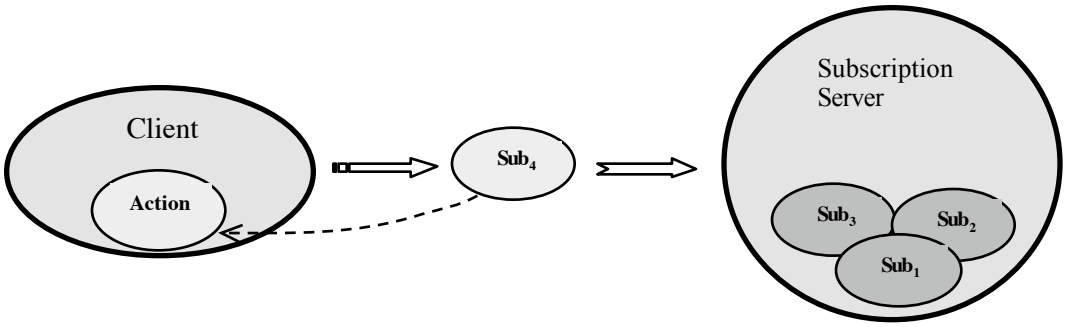


Fig.30(a): Subscription registration

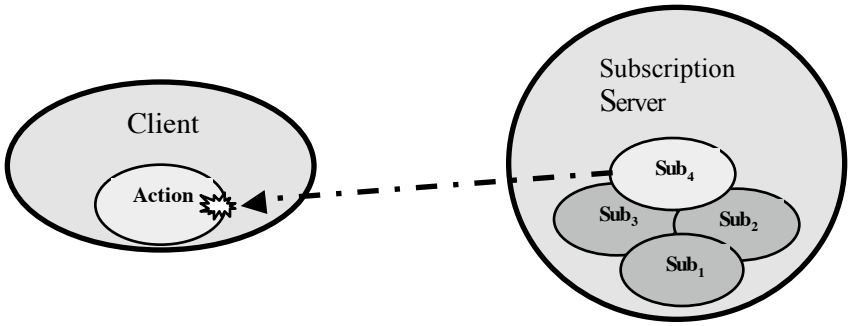


Fig.30(b): Client notification regarding interest satisfaction

4.2 Agent Engine

The Agent Engine represents the ***logic tier*** of ICDM's underlying three-tier architecture. Existing as a client to the services of the ***information tier*** (i.e., access, query, subscription, and persistence) the Agent Engine is capable of both obtaining and injecting information. Architecturally, the Agent Engine consists of an agent server capable of serving collections of agents (Fig.27). These collections, or Agent Sessions, exist as self-contained, self-managing agent communities capable of interacting with the ***information tier*** to both acquire and inject information.

For the most part, the exact nature of the agents and the collaborative model employed is left to the application specification. However, regardless of the types of agents contained in an Agent Session, agent activity is triggered by changes in application information. This information may take the form of global objects managed by the ***information tier*** or local objects utilized in agent collaboration, which are managed by the Agent Session itself. Regardless of whether agents are interacting with the ***information tier*** or each other, interaction takes place in terms of objects.

This again illustrates the degree to which an object representation is preserved as information is processed throughout the application environment.

4.3 Agent Session Configuration

The grouping of agent analyses into heterogeneous collections of agents allows for a number of interesting configurations. These configurations determine the size, number, and individual scope of the agent sessions. While a wide variety of Agent Session configurations exist, we have found considerable success in formulating such a configuration based on two primary criteria.

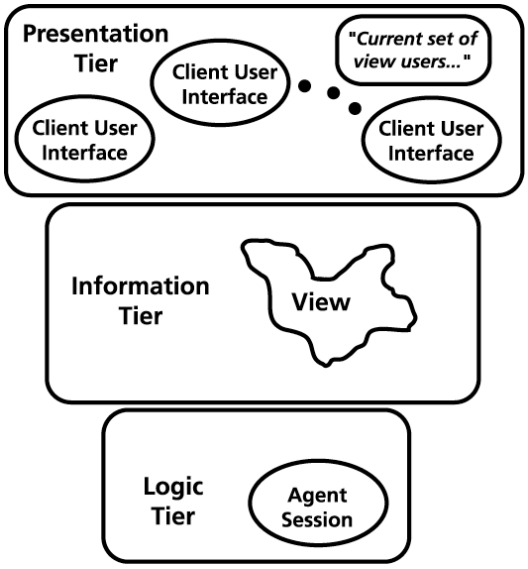


Fig.31: Multiple users can interact with a ***view*** that is analyzed by one Agent Session

The first criterion introduces the notion of a ***view***. A ***view*** is a conceptual perspective of reality. In other words, a ***view*** can be thought of as a single investigation into solving a problem whether it is based on fact or speculation. For example, a ***view*** may describe events and information relating to what is actually occurring in reality. Yet, another ***view*** may describe an alternative or desired reality. An illustration of this approach can be observed in the Integrated Marine Multi-Agent Command and Control System (IMMACCS) developed by the CADRC and CDM Technologies for the Marine Corps (Pohl et al. 1999). IMMACCS uses a single ***view*** to represent the information and events occurring in the battlespace. In a similar manner, IMMACCS employs any number of additional views to represent hypothetical investigations to determine suitable strategies for dealing with potential events or circumstances.

Regardless of use, however, there is a one-to-one correspondence between a conceptual ***view*** and an Agent Session (Fig.31). This means that independent of exactly which version of reality a ***view*** represents, there exists a dedicated Agent Session providing users of that ***view*** with agent-based analysis and decision-support. Each agent of a particular Agent Session deals only with the ***view*** associated with its Agent Session. Organizing information analysis in this manner allows for an efficient and effective means of distinguishing information and activities relating to one ***view*** from those pertaining to another. Unless prompted by user intervention (i.e., user-directed movement of information between views), each set of information is disjoint from the other.

The second configuration criterion that we currently subscribe to determines the number and nature of agents contained in an Agent Session at any point in time. Our decision-support applications typically utilize a variety of agent types. Three of these agent types include Domain Agents, Object or Mentor Agents, and Mediator Agents (Pohl 1995). Service-oriented Domain Agents embody expertise in various application-relevant domains (i.e., weapon selection and deconfliction for tactical command and control, tidal dynamics and trim and stability for ship load planning, and so on). The collection of Domain Agents populating an Agent Session at any point in time determines the variety of domain specific perspectives and the analytical depth available during the analysis of the associated ***view***. Under the configuration scheme utilized in most of our decision-support applications, users can add or remove these domain perspectives in a dynamic fashion.

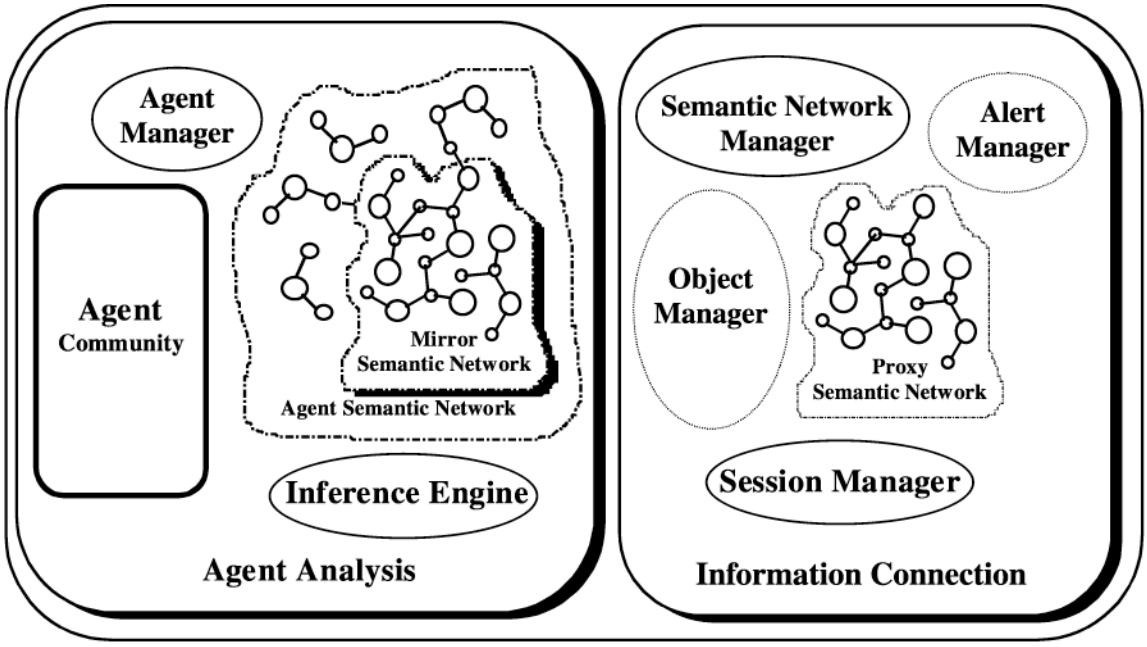


Fig.32: Agent Session architecture

Object or Mentor Agents, on-the-other-hand extend the notion of high-level informational representation by essentially agentifying information through empowering information objects with the ability to act on their own behalf. Both human users and other agents as needed can initiate this agentification of information into Object Agents.

In an attempt to resolve conflicts arising between collaborating agents, Mediation Agents may be employed as third party mediators. It is the goal of these mediators to bring about consensus among agents that may have reached an impasse.

Under the current ICDM model each of these agent contingents is dynamically configurable by both the user(s) and the system itself. This approach to Agent Session configuration promotes the notion of offering assistance in the form of dynamically configurable tools rather than predefined solutions (Pohl 1997).

4.4 Agent Session Architecture

Architecturally, an Agent Session consists of several components including the Semantic Network and Semantic Network Manager, Session Manager, Inference Engine, and Agent

Manager (Fig.32). These components operate in an integrated fashion to maintain a current information connection between the agents residing in the Agent Session and the associated ***view*** described in the ***information tier***.

4.5 Semantic Network

The Semantic Network consists of a collection of two sets of application specific information objects. The first set is used to support local collaboration among agents. Depending on the specific collaborative model employed, agents may use this local Semantic Network to propose recommendations to each other or request various services. This information is produced and modified by the agents and remains local to the Agent Session.

The second set of information is a sort of duplicate, mirror image of the ***view*** information stored in the ***information tier***. In actuality, this information exists as a collection of object-based interfaces allowing access to ***view*** information stored in the Information Server. Such interfaces are directly related to the application's ontological model. In other words, these interfaces, or proxies (Mowbray and Zahavi 1995), are represented in terms of the objects described in the information object model.

Through these interfaces, Information Server clients have the ability to access and modify objects contained in the ***information tier*** as though they are local to the client's environment. All communication between the object interfaces and their remote object counterparts is encapsulated and managed by the Information Server and completely transparent to the clients. This is a fundamental feature of distributed object servers on which the Information Server is based (Orfali et al. 1996).

4.6 Semantic Network Manager

As the primary manager of the two sets of information described above, the Semantic Network Manager focuses the majority of its efforts on the management of the bi-directional propagation of information between Information Server proxies and an equivalent representation understandable by the Inference Engine. Such propagation is accomplished through employing an Object Manager. The purpose of this manager is to essentially maintain mappings between the object proxies and their corresponding Inference Engine counterparts. The necessity of this mapping reveals a limitation inherent in most distributed object server and inference engine facilities. Most facilities supporting one of these two services require control over either the way client information is represented or the manner in which it is generated. This is due to the fact that both facilities require specific behavior to be present in each object they process. Examples of such facilities include IONA's ORBIX distributed object server (IONA 1996) and NASA's CLIPS inference engine (NASA 1992). Both of these facilities suffer from this limitation. Nonetheless, this dilemma can be solved through the use of an intermediate object manager that maintains mappings between the two sets of objects.

An additional responsibility of the Semantic Network Manager deals with the subscriptions, or interests, held on behalf of the agent community. That is, the Semantic Network Manager is responsible for maintaining the registration of a dynamically changing set of information interests held on behalf of the Agent Session agents. In addition, the Semantic Network Manager is responsible for processing notification(s) when these interests are subsequently satisfied. Such processing includes the propagation of information changes to the agent community that may in turn trigger agent activity.

To perform these two interest-related tasks the Semantic Network Manager employs the services of the Alert Manager. The Alert Manager provides an interface to the Subscription Server facility and is available to any ***information tier*** client wishing to maintain a set of information interests. Employment of the Alert Manager by subscribers has two distinct advantages. First, clients are effectively decoupled from the specifics of the subscription interface. This allows the same application client to be compatible with a variety of object server implementations. Second, the Alert Manager interface allows subscribers to effectively decompose themselves into a dynamic collection of thread-based interest clients (Lewis and Berg 1996). In this respect, the Alert Manager extends the monolithic one-to-one relationship between the Subscription Server and its clients into one that supports a one-to-many relationship. Such decomposition of functionally related behavior into lightweight processes promotes the concepts of multi-processing in conjunction with resource conservation.

4.7 Inference Engine

The Inference Engine provides the link between changes occurring in the Semantic Network and agent activation. Recall that agent activation can occur when a change in the Semantic Network is of interest to a particular agent. In such a case, the Inference Engine, having knowledge of specific agent interests in addition to changes occurring in the Semantic Network is responsible for activating, or scheduling the action(s) the agent wishes to execute. This activation list forms the basis for the Agent Manager to determine which agent actions to execute on behalf of the currently scheduled agent.

4.8 Agent Manager

The Agent Manager is responsible for the management of the agent community housed in an Agent Session. This management includes the instantiation and destruction of agents as they are dynamically allocated and de-allocated to and from the agent community. In addition, the Agent Manager is responsible for managing the distribution of execution cycles allowing each agent to perform operations. Disbursement of execution cycles occurs in a round-robin fashion allowing agent analysis to be evenly distributed among relevant agents. Whether or not an agent utilizes its allotted cycles depends on whether it has any tasks or actions to perform.

4.9 Session Manager

As the overall manager of the Agent Session environment the Session Manager has two main responsibilities. The first of these responsibilities focuses on the initialization of each of the other Agent Session components, upon creation. When an Agent Session is created in response to the creation of a ***view***, the Session Manager is the first component to be activated. Once initialized, the Session Manager activates the Semantic Network Manager and Inference Engine. Continuing its efforts, the Session Manager then activates the Agent Manager. Upon startup, the Agent Manager initializes itself by allocating an appropriate initial set of agents. Depending on the application specifics, these agents may in turn perform a series of initial queries and subscriptions that will eventually propagate to the ***information tier*** via the Semantic Network Manager.

5. ICDM in Practice: *IMMACCS*

5.1 The Object Model: IMMACCS as an Example

Fundamental to the ICDM software development framework is the representation within an ICDM-based application of ***information*** rather than data. In this regard data are defined as numbers and words without relationships. Information, on the other hand, is defined as data with relationships to provide at least some minimum level of context. Accordingly, information processed within an ICDM-based application is described in terms of objects having attributes, behavior, and relationships to other objects. Collectively, these descriptions form the information object model (i.e., ontology) of the application.

The object model that was developed for the ICDM-based application, IMMACCS, to represent the characteristics and relationships of the information expected in the urban battlespace environment is significant even beyond the scope of IMMACCS. A major achievement of this project has been to take an object model (i.e., ontology) through a complete development cycle, creating the necessary tools along the way to highly automate the process. This capability allows for the development of an object model using a standard graphical methodology, the Unified Modeling Language (UML), and directly produce final application code.

5.1.1 IMMACCS Object Model Goals

IMMACCS was conceived as a distributed, open architecture, command and control (C2) system to assist military commanders under battle-like (i.e., execution) conditions when dynamic information changes, complex relationships, and time pressures tend to stress the cognitive capabilities of decision-makers and their staff. To meet the specifications of a military command and control system IMMACCS was required to:

1. Support multiple applications required for US Marine Corps command and control, with particular regard to:
   - 1.1 Situation awareness (i.e., common operational picture).
   - 1.2 Adaptability for future requirements.

2. Enable decision-support through collaboration, by supporting:
   - 2.1 Human decision-making through tailored information presentation.
   - 2.2 Agent decision assistance through knowledge capture in complex information structures and logical rules.

3. Object-oriented information representation, to provide:
   - 3.1 A common information structure for enabling multiple source integration.
   - 3.2 Unambiguous representation.

As further background it should be mentioned that the IMMACCS application was designed and implemented in concert with its military users as an integral component of experiments conceived by the Marine Corps Warfighting Laboratory (Quantico, Virginia) to test emerging concepts in military command and control. It was field tested as the command and control

system of record during the Urban Warrior Advanced Warfighting Experiment conducted by the US Marine Corps in Monterey and Oakland, California, March 12 to 18, 1999, during a live fire Limited Objectives Experiment (LOE-6) held at Twentynine Palms, California, in March, 2000, and again during the Kernel Blitz military exercise in July 2001. (Design and development responsibilities for IMMACCS were assigned as follows: overall design concept, Agent Engine, Object Model, and Object Browser (CADRC, Cal Poly, San Luis Obispo, California); Shared Net and Object Instance Store (Jet Propulsion Laboratory, Pasadena, California); objectified infrastructure (Navy Research Laboratory, Stennis Space Center, Missouri); 2-D Viewer and Backup System (SRI International, Menlo Park, California); Translator(s) for external (i.e., legacy) applications and System Engineering Integration (SPAWAR Systems Center, San Diego, California.)

5.1.2 IMMACCS Object Model Structure

The IMMACCS Object Model (IOM) is subdivided into packages of classes, referred to as domains. Each domain models a different aspect of the battlespace.

1. **Base Domain**
   - 1.1 ***View:*** A logical container of IMMACCS objects that may have associated regions (geographic); used to segregate objects according to usage (e.g., execution, planning/shaping, etc).
   - 1.2 ***BaseObject:*** A base class providing common characteristics; provides link to optional narratives (i.e., textual descriptions); and, includes optional resource specifications (e.g., reach-back information).
   - 1.3 ***Physical:*** Objects that have geographic location; may represent un-validated observations that may be correlated into a validated assessment.
   - 1.4 ***Track:*** Physical objects that can be tracked (e.g., objects that are mobile) and may be located in (i.e., associated to) places or environments.
   - 1.5 ***Agent:*** Used for the representation of agents; exposes agent decision-support characteristics (e.g., alerts, agent type, etc.).
   - 1.6 ***Information:*** Descriptive objects that provide additional information for execution (i.e., operation) and planning/shaping (i.e., tactical).
   - 1.7 ***Alert:*** Used for informational feedback from agents.

2. **Entity Domain** (includes corporeal objects that may be tracked)
   - 2.1 ***Unit:*** Organizational entities; may have members and assets.
   - 2.2 ***MilitaryUnit:*** Units with defining characteristics specific to military organizations; extensive support for specialized ground units (based on MIL-STD 2525b).
   - 2.3 ***Person:*** Individual personnel (may be members of unit(s)).

3. **Asset Domain** (includes non-corporeal objects that may be tracked)
   - 3.1 ***Platform:*** Transport assets; may have embarked tracks; may have fuel supply; and, support for specialized platforms (based on MILSTD 2525b).

3.2 ***Equipment:*** Specialized weapons and sensors.

4. **Supply Domain** (includes consumable supplies that are used by units, personnel, platforms, weapons, etc.; may include quantity (including quantity thresholds for supply status assessment)).

5. **Environment Domain** (includes the static (non-tracked) environment)
   - 5.1 ***Artificial Environment:*** Specialized non-natural environments.
   - 5.2 ***Building:*** Artificial structures that have internal spaces.
   - 5.3 ***Infrastructure:*** Supporting structure (e.g., roads, bridges, utilities).
   - 5.4 ***Complex:*** Facilities that may have structures.
   - 5.5 ***BuiltupArea:*** Populated areas that may have structures and complexes.
   - 5.6 ***Natural Environment:*** Specialized natural environments (e.g., land, littoral, etc.).

6. **Tactical Domain** (includes classes defining tactical information used for planning and shaping the battlespace; may have associated operational information; and, may be geographically located)
   - 6.1 ***Overlay:*** Container of tactical information objects (including other overlays).
   - 6.2 ***Overlay Object:*** Tactical objects that have geographical extent; supports graphical characterization for visualization.
   - 6.3 ***Control Measures:*** Objects that have influence on battlespace tactics; may have associated environment; and, may have associated affected tracks.
   - 6.4 ***Mobility:*** Tactical objects that constrain mobility.
   - 6.5 ***Maneuver:*** Control measures that influence battlespace maneuvers; may have associated tasks.
   - 6.6 ***Fire Support:*** Control measures used in fires planning and shaping.
   - 6.7 ***Combat Service Support:*** Control measures used in logistics planning and shaping.

7. **Operation Domain** (includes classes defining operational information used during execution; 'fire' information and support for field fires requests (e.g., call for fire, etc), reports, and orders)
   - 7.1 ***Intel Information:*** Field Intel reconnaissance reports.
   - 7.2 ***Combat Service Support Information:*** Field logistics reports and requests (e.g., medical evacuation, resupply, etc.).
   - 7.3 ***Combat Information:*** Field combat reports.

5.1.3 Model Maintenance

The Unified Modeling Language (UML) was and continues to be used for model development, and the Extensible Markup Language (XML), XML Meta-Data Interchange (XMI), and XMI-UML Meta-Model, are used for model storage. In addition, IMMACCS utilizes the Object Constraint Language (OCL) for the definition of specialized presentation mappings.

5.1.4 Artifact Generation

All artifacts are generated from XMI-UML in several formats, namely: CORBA Interface Definition Language (IDL); Object Management Layer (OML) Class Properties; CLIPS Class Definitions; and, model documentation (Fig.33).

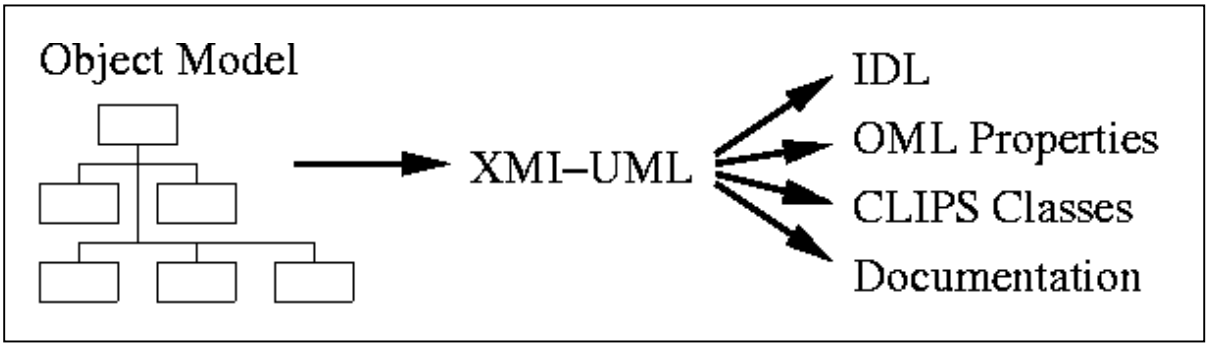


Fig.33: Generated Object Model artifacts

5.2 The Object Management Layer (OML)

As a component of the ICDM Toolkit framework, OML is designed as a class library that provides general functionality for complete life-cycle management of objects, their attributes (i.e., characteristics) and associations (i.e., relationships). Interaction with object instances is simplified through the use of simple strings with attribute value constraints handled internally. Association management is also provided internally alleviating the requirement and complexity of ensuring referential integrity by the using application. Management of interests is also provided, implemented internally, and exposed to using applications through the standard Java event model. Additionally, support is provided for accessing multiple servers simultaneously and transparently.

The primary application of this library is for clients that have little or no prior knowledge of the object domain model(s). Internally, the required management and information is provided through runtime reflection and properties. Good examples of such applications are user interfaces in which a hard-coded notion of the domain is expensive to develop and manage.

5.2.1 OML Capabilities and Classes

As a class library that greatly facilitates the development of client software for interaction with domain objects, OML offers the following capabilities:

1. A client-side programmer's API that supports generic run-time object discovery and interaction constraint, including:

1.1 Association management.

1.2 Attribute value string conversion.

1.3 Object creation, deletion, and modification transactions.

1.4 Event notification for object creation, deletion, and modification on instance and class based interests.

2. Transparent interaction with multiple server interfaces (i.e., each tied to a high level domain) with plug-in API classes.

   2.1 Object Server API (i.e., allows interaction with distributed / persisted objects).

   2.2 Null Server API (i.e., allows interaction with local objects).

   2.3 Object Environment Server API (i.e., provides extra functionality over the Null Server API to simplify derived attribute implementation and interaction with objects that have internal behavior).

3. OML classes.

   Template (domain object class management)

   - forms template from class inheritance for object creation;
   - supports class introspection through Java reflection/class properties;
   - supports object construction and destruction;
   - manages class-based interests.

   POW (proxy object wrapper)

   - constrains domain object instance interaction;
   - passes attribute values as strings and enforces type constraint;
   - manages association role values as string references and enforces integrity.

   Attribute and subclasses (attribute constraint management)

   - supports type introspection;
   - supports type/string conversion;
   - allows the use of specialized plug-in classes to override provided classes.

5.3 The IMMACCS Agent Engine Example

The particular Agent Engine configuration that was developed for the ICDM-based IMMACCS application, monitors and reacts to changes in the characteristics and relationships of the information expected in the urban battlespace environment. Depending on the conditions existing in the battlespace, the IMMACCS Agent Engine (IAE) may create, modify, or delete objects. The agents collaborate with the operators through the creation and modification of Alert objects. Alert objects contain messages and associations to other objects in the battle-space that aid the operator in the decision-making process. Alerts are displayed through the Agent Status Panel

(ASP), a component that can be integrated into the IMMACCS Graphical User Interface (IGUI) or operate independently. It is through the ASP that the operator interacts with the IAE.

Within the IMMACCS Agent Engine there exist two types of agents. Static service agents are created at the beginning of an Agent Session and generally monitor the battle-space reacting to changes in objects that fall within their particular domain of expertise. Service agents are open to collaboration with any operator that has access to IMMACCS. Dynamic mentor agents are not necessarily created at the beginning of an Agent Session, rather their creation is dependent upon the presence of their operator in the battlespace. As an operator accesses IMMACCS, a dynamic mentor agent is created and monitors the battle space for its operator until the operator leaves IMMACCS, at which time the dynamic mentor agent is destroyed. While active in the battle space the dynamic mentor agent monitors the battle space for conditions relevant to its operator, collaborating only with its operator.

5.3.1 IMMACCS Agent Capabilities

Mentor Agent – a dynamic agent that serves the interests of only the operator for which it was created.

- An alert is deemed to be of interest if the operator's track is the causing or affected object of a service agent alert, a subordinate's track is the causing or affected object of a service agent alert, or if an asset owned by the operator is the causing or affected object of an agent alert.
- Once an alert has been viewed it is automatically acknowledged and cleared from the Agent Status Panel.
- When the operator logs out of IMMACCS the Mentor agent is destroyed.

Monitor Agent – a dynamic agent created by the operator that monitors specific objects in the battle space as chosen by the operator.

- If an object designated by the operator becomes the causing or affected object of an agent alert, the operator is notified.
- Once an alert has been viewed it is automatically acknowledged and cleared from the Agent Status Panel.
- When the operator logs out of IMMACCS the Monitor agent is destroyed.

NAI Agent – a static service agent that monitors all Named Areas of Interest in the battle space.

- A NAI is established by creating a polygon with three or more sides at a specific location in the battle space. Courses of Action (COA) may also be associated to a NAI.
- When a HOSTILE or NEUTRAL track enters a NAI, the NAI agent creates an alert including any Courses of Action (COA) that were established during the planning process.
- When the HOSTILE or NEUTRAL track leaves the NAI, the NAI agent deletes the alert.

TAI Agent – a static service agent that monitors all Targeted Areas of Interest in the battle space.

- A TAI is established by creating a polygon with three or more sides at a specific location in the battle space. COA may also be associated to a TAI.
- When a HOSTILE or NEUTRAL track enters a TAI, the TAI agent creates an alert including any Courses of Action that were established during the planning process.
- When the HOSTILE or NEUTRAL track leaves the TAI, the TAI agent deletes the alert.

Decision Point Agent – a static service agent that monitors Decision Points in the battle space.

- A Decision Point is established in the battle space by selecting a location and identifying a tolerance (or radius) around that location.
- When a FRIENDLY track enters the tolerance (or radius) of a Decision Point, the Decision Point agent creates an alert including any Courses of Action that were established during the planning process.
- When the FRIENDLY track leaves the Decision Point, the Decision Point agent deletes the alert.

Blue on Blue Agent – a static service agent that monitors the battle space for friendly fire conditions.

- When a FRIENDLY unit exists in the battle space and a CFF target location comes within 500 meters of its location, an alert is created.
- When the distance between the FRIENDLY unit and the CFF target location exceeds 500 meters or the CFF is canceled, the alert is deleted.

Sentinel Agent Capabilities – a dynamic mentor agent that monitors the vicinity of its operator and generates an alert on HOSTILE track encroachment.

- When an operator accesses IMMACCS, a Sentinel agent is created for that operator.
- When a HOSTILE track encroaches within 4000 meters of the operator, an alert is created.
- When the HOSTILE track moves to a distance greater then 4000 meters, the alert is deleted.

Intel (or RADAR Missile Launcher) Agent – a static service agent that monitors non-FRIENDLY missile launcher or RADAR tracks in the battle space.

- When a non-FRIENDLY missile launcher or RADAR track exists in the battle space and its status is listed as “passive”, an alert is created calling attention to the existence of a high value target, suggesting that a Call for Fire be submitted.

- When a non-FRIENDLY missile launcher or RADAR track exists in the battle space and its status is listed as "active" an alert is created calling attention to the existence a status of a high value target while simultaneously submitting a Call for Fire.
- When the status of the non-FRIENDLY track has been degraded from "active" or "passive", the alert and CFF is deleted.

Engagement (or Enemy) Agent – a static service agent that monitors the proximity of FRIENDLY and HOSTILE tracks relative to each other in the battle space.

- When a FRIENDLY track comes within range of a HOSTILE track's typical organic fires assets, an alert is created.
- The rules of thumb that determine typical organic weapons and subsequently the ranges are based on the type of combat unit the track represents as well as the track's echelon.
- The creation and association of actual weapons for a HOSTILE track can override the rules of thumb for typical organic weapons.
- When a FRIENDLY track moves outside of the range of a HOSTILE track's weapons, the alert is deleted.

ROE Agent – a static service agent that monitors simplified Rules of Engagement (ROE) within the battle space.

- When a CFF target location is within 300 meters of a NEUTRAL civilian track, an alert is created.
- When the NEUTRAL civilian track moves farther then 300 meters from the CFF target location or the CFF is deleted, the alert is deleted.

CASEVAC (or MEDEVAC) Agent – a static service agent that monitors the battle space for CASEVAC requests.

- When a CASEVAC request is created in the battle space the CASEVAC agent identifies the nearest FRIENDLY track that may be of assistance.
- The search is limited to FRIENDLY MEDEVAC or Search and Rescue rotary wing tracks, Medical Support Unit tracks, or Ground Vehicles.
- When a CASEVAC request is removed from the battle space, the alert is deleted.

DBMA Agent – a static service agent that monitors the battle space tracking the Dynamic Battle Management Area (DBMA) of FRIENDLY tracks.

- The longest effective range of any indirect fires weapons associated to a FRIENDLY track determines its DBMA. The DBMA can be thought of as analogous to a radius of influence.
- When a FRIENDLY track exists in the battle space and has an indirect fires weapon associated to it, an alert is created.

- When additional indirect fires weapons are associated to a FRIENDLY track that alters the longest effective range for that FRIENDLY track, the alert is revised.
- When all of a FRIENDLY track's indirect fires weapons are disassociated, the DBMA is set to zero and an alert is created notifying the operator of this condition.
- Once an alert for the zero DBMA condition is acknowledged, the alert is deleted.

Fires Agent – a static service agent that responds to a Call for Fire (CFF) message by selecting the best weapon with which to engage the CFF target.

- To select the best weapon, the Fires Agent references an "appropriateness" matrix for weapons available to engage the type target, considers the distances from each weapon to the target, includes the effective casualty radii (ECR) and circular errors of probability (CEP) of the weapons, and considers any tracks positioned along the line of sight from the weapon to the target.
- The weapon are ranked and presented to the operator based on these parameters.

Logistics Agent – a static service agent that monitors selected supplies for FRIENDLY tracks in the battle space.

- Monitors the fuel level in friendly or neutral tracks creating a YELLOW alert when the fuel level falls below 50% and a RED alert when the fuel level falls below 25%. Potential supply points, based on track location, are also identified.
- Monitors the supply level for friendly tracks creating a YELLOW alert or a RED alert when the supply levels falls below preset levels. Potential supply points, based on track location, are also identified.

Hazard Agent – a static service agent that monitors the battle space for nuclear, biological, or chemical events.

- When a non-Hostile track enters the area defined by a NBC object (a type of Overlay object) an alert is created.
- As non-Hostile tracks continue to enter, or as the tracks leave the area defined by the NBC object the alert is updated to reflect these changes.
- When all the non-Hostile tracks have left the area defined by the NBC Object, the alert is deleted.

6. Some Software Design and Development Considerations

This Section discusses aspects of the design and development of ontology-based applications that are related to the dual objectives of achieving ***interoperability*** and creating applications that do in fact satisfy the needs and desires of the end-users. 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*** (rather than data) 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.

Ontologies are focused on the utilization of data for functional (i.e., operational) purposes. They provide the context within which software agents are able to provide meaningful assistance to human users in the interpretation of data changes and the decision-making process that typically follows such interpretation. It is therefore of critical importance that the end-users of an ontology-based application be involved in the design of the ontology. The necessary knowledge acquisition process is greatly facilitated by the use-case methodology. Use-cases may be in textual or flow diagram form and describe the desired behavior of a proposed application in responding to user requests.

6.1 Ontology Approach to Semantic Interoperability

An ontology can be characterized as an explicit specification of a conceptualization. The term is borrowed from philosophy, where an ontology is a systematic account of existence. For a software application, what "exists" is that which can be represented. When the information and knowledge of a domain is represented in a declarative formalism, the set of objects that can be represented is called the universe of discourse. This set of objects, and the describable relationships among them represents all the information and knowledge that can be known in the context of the applications that employ them. In such an ontology, definitions associate the names of entities in the universe of discourse (e.g., classes, relations, functions, or other objects) with human-readable text describing what the names mean, and formal axioms that constrain the interpretation and well-formed use of these terms.

In terms of semantic interoperability, an ontology defines the vocabulary with which queries and assertions are exchanged among applications. Ontological commitments are agreements to use the shared vocabulary in a coherent and consistent manner. The applications sharing a vocabulary need not share a knowledge base; each knows things the other does not, and an application that commits to an ontology is not required to answer all queries that can be formulated in the shared vocabulary.

An Interface Domain Ontology is an ontology specifically geared towards interfacing multiple domain specific software systems. The concepts in an Interface Domain Ontology can be organized in a hierarchical structure of three layers as shown in Fig.34. The Upper Level Ontology describes generic concepts such as *process*, *agent*, *set* and *goal*, while the Lower Level Ontology describes elementary concepts such as *SSN*, *NEC*, *cost* and *Internet address*. Generally, for two cooperating partners, it is relatively easy to reach a consensus on the concepts of these two parts especially if they both operate within a common overarching domain such as the US Navy, which provides for common terms and concepts. The difficult section is the Application

Level Ontology. The concepts defined at this level depend strongly on the specific application domains to be encompassed by the interface, which dictates the kind of problems to be addressed, the method used to solve them, and the underlying technology which often contaminates the model of a particular application domain. Typical concepts in this layer are 'supply requisition', 'maintenance action', 'efficiency rating', and ‘reliability index’.

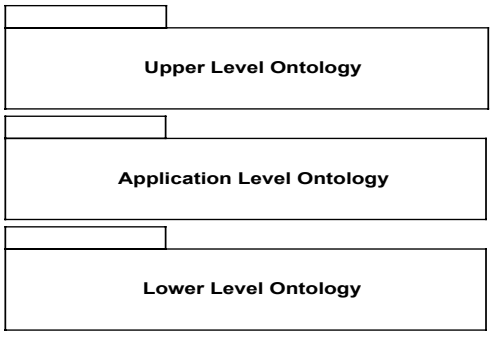


Fig.34: Ontology semantic levels

6.1.1 Interoperability Example

In order to better understand these concepts a simple example is in order. This example considers the relationships between supply and maintenance activities and the corresponding information systems that support them. Assume that the supply and maintenance systems were initially developed in complete isolation from one another with the respective goals of automating the internal processes of the supply and maintenance departments. These processes were based on the flow of standardized paper forms through the various sections of the two organizations. The forms are delivered to the in-box of a particular section whose members typically perform some real world action that is recorded on the original form or on a new form resulting from the action. These records are then placed in the out-box of the section to begin the next leg of the journey specified by the department process. The automation provided by the information system for each of these two activities essentially mirrors the respective manual processes but replaces the physical entities of the process such as paper forms and in/out boxes with virtual representations that exist only within the confines of a computer.

Additionally, assume that automation provided by these systems is internal to the corresponding activity, which requires the generation of physical artifacts to interface with dependent activities. The maintenance activity produces paper-based supply requisitions for delivery to the supply activity, which acts to fulfill the request eventually producing a paper-based shipment order that is returned to the maintenance activity indicating the requested physical parts that are to be delivered. Maintenance system users that wish to know the status of their shipment would contact Supply Department personnel by phone. Supply personnel would then query the system to provide a verbal status report to the Maintenance Department caller.

Internal to each of these activities, many other types of documents, and tables are employed to manage them such as maintenance and delivery schedules, shipping rates, and trouble-shooting protocols. In ontological terminology these two different sets of entities and artifacts are known as domains, which this example further specifies as the Maintenance Domain, and the Supply Domain. For the purposes of this discussion these domains can be thought of as having three layers. The semantic layer describes the structure of the domain entities and the relationships between them that together comprise a model for representing the corresponding real world problems within the computer. This is a conceptual layer that is somewhat independent of the

physical implementation.

The data or information layer contains specific instances of the semantic layer entities linked together in a manner that describes the complete contextual state of a given domain. This is a physical layer that requires a specific implementation paradigm. The entities and relationships defined in the conceptual layer may manifest themselves as linked class instances in an object-oriented paradigm or as related table records in a relational paradigm. The Agent layer contains the domain users and software agents that leverage the information layer to perform useful tasks. The data and information that flows between these domains can be called the Maintenance and Supply Interface Domain or just Interface Domain for short. The Interface Domain is the focus of this example and this Section.

As both the example systems evolve and the internal automation is nearing completion or is at least well understood, it is natural that they look to extend the automation across the activity boundaries. Subsequently this Section will use the domains and layers just described to present three successive levels of system to system interaction: Data Level System Interface, Information Level System Interface, and Information Level System Interoperability to characterize different ways in which this automation could be realized.

6.1.2 Data Level System Interface

A data level system interface is characteristic of most of the interfaces between data-centric systems at the present time. As the information to be exchanged enters into the interface domain, it looses most context because this type of interface views each exchange as unrelated chunks of data. In this case assume supply system developers are responsible for developing an interface to the maintenance system to periodically pull supply requisitions and the maintenance system developers are responsible for developing an interface to obtain supply shipment status information as requested by maintenance system users. Each group of developers designs a record set for the required data, which together define the interface specification (Fig.35).

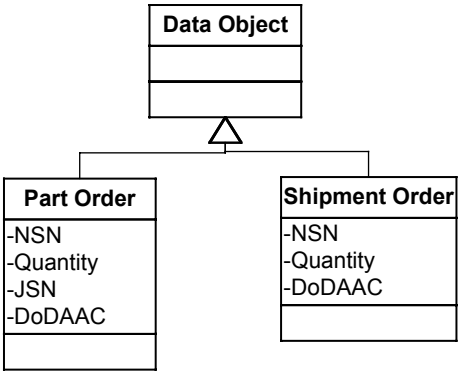


Fig.35: Data Level Interface specification

Although the focus of the interface is the exchange of data rather than semantic content, there is still an ontology associated with the interface. The explicit record specifications (Fig.35) represent the application level of the interface domain ontology for the example interface. Most of the attributes found in the record specifications are referenced to entries in the Defense Data Dictionary System (DDDS), which in this case serves the role of the Lower Level Interface Domain Ontology. By marking up the interface attribute definitions in terms of DDDS entities one can easily determine that NSN is the National Stock Number, JSN is the Job Serial Number, and DODAAC is the Department of Defense Activity Address Code. One can reference these in

other documentation and data specifications to further ascertain the conceptual meaning associated with them.

While there is no explicit Upper Level Domain Ontology there is an implicit one, which greatly assists developers in finding the common ground to implement this interface. This upper level ontology is an implicit artifact of the standardized processes of the underlying domain, the Department of the Navy in this case, which defines common conceptual entities such as a Maintenance Action Form and a Supply Requisition Form. These common conceptual entities in combination with the attributes defined in the DDDS provide the developers of the two information systems a common vocabulary with which to discuss, design, and develop specific interfaces between their respective systems.

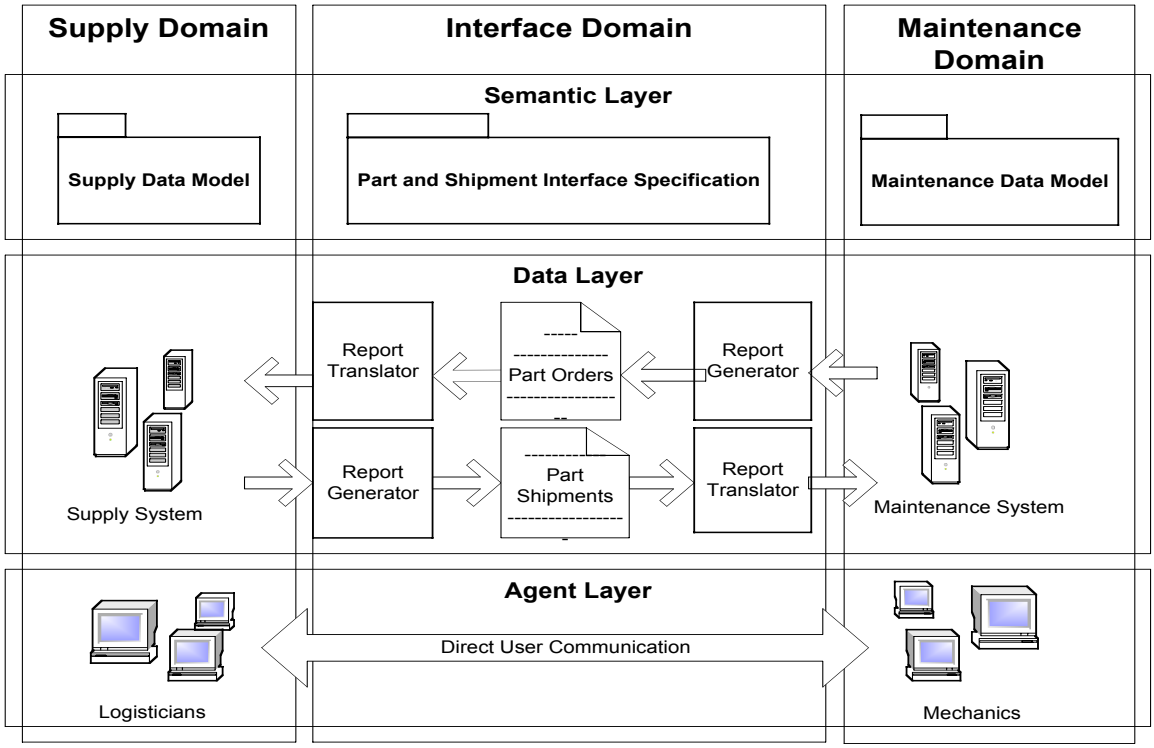


Fig.36: Data Level System Interface

At this point the Semantic Layer of the Data Level System Interface has been defined and is depicted as the Part and Shipment Interface Specification in Fig.36. The Semantic Layer depicts the internal data models of the Supply and Maintenance domains as well but these fall short of an ontology or even of a specification because they are considered the private proprietary property of the individual organizations responsible for developing the respective systems. It is also likely that these internal models are more focused on the individual forms and tables that users want to appear on their screens rather than on the underlying semantic entities the screens were designed to display information about. This makes it difficult to understand the models outside the context of the applications they were designed to support. While the interface specification appears well defined, the context from which the data are extracted on one end of the interface and then inserted on the other is not addressed by the specification at all.

The Data Layer of the Data Level System Interface realizes the interface specified in the semantic layer. In the case of this example, the maintenance system developers must be responsible for

developing the report generator code that pulls the requisite data from the context provided by the maintenance data model to generate the list of part orders that constitutes the interface to the supply system and for developing the report translator code that translates the part shipment interface records into the context of the maintenance data model.

Similarly, the supply system developers must be responsible for developing the report generator code that pulls the requisite data from the context provided by the supply data model to generate the list of part shipments that constitutes the interface to the maintenance system and for developing the report translator code that translates the part order interface records into the context of the supply data model. Neither group of developers is really sure how the other group generated the data they need nor are they sure of what the other group does with the data they generate as they have no visibility into each others data models. The report translators and generators depicted on the figure are representative of these hidden context shifts into hidden proprietary data models.

As data level system interfaces such as that described in this example go to the field problems often arise. Since developers are often guessing about the context on either end they often do not get it quite right. This requires the logisticians and mechanics that use the systems to perform work-arounds in the field to accomplish such additional filtering or hand tweaking to the records generated from the external system. Users of these types of data-centric systems are used to this sort of data massaging and their systems are well suited to this as the meanings of fields in a data level system are easy to use in locally defined ways. Of course this further complicates the problem, as these sorts of local modifications require local tweaks to the interfaces and ultimately produce an interface that marginally accomplishes the intended purpose.

6.1.3 Information Level Interface

An information level interface differs from a data level interface in several ways. Primary among these is the requirement for the systems being interfaced to be information-centric rather than data-centric. Information-centric systems are based on explicit ontologies that model the underlying semantic entities of the domain rather than the data crunched by the currently favored domain processes or displayed on the screens of particular applications.

The developers of an information level interface consider all the information to be exchanged (parts and shipments in this case) in a singular context, which not only relates the entities to be exchanged to each other but to the context in which the entities are related at both ends of the interface. This is show in the Semantic Layer of the Information Level System Interface depicted in Fig.37 by an Interface Ontology that overlaps into the Supply and Maintenance Domains. The Interface Ontology is marked up in terms of the shared (public) Supply and Maintenance ontologies and vice versa.

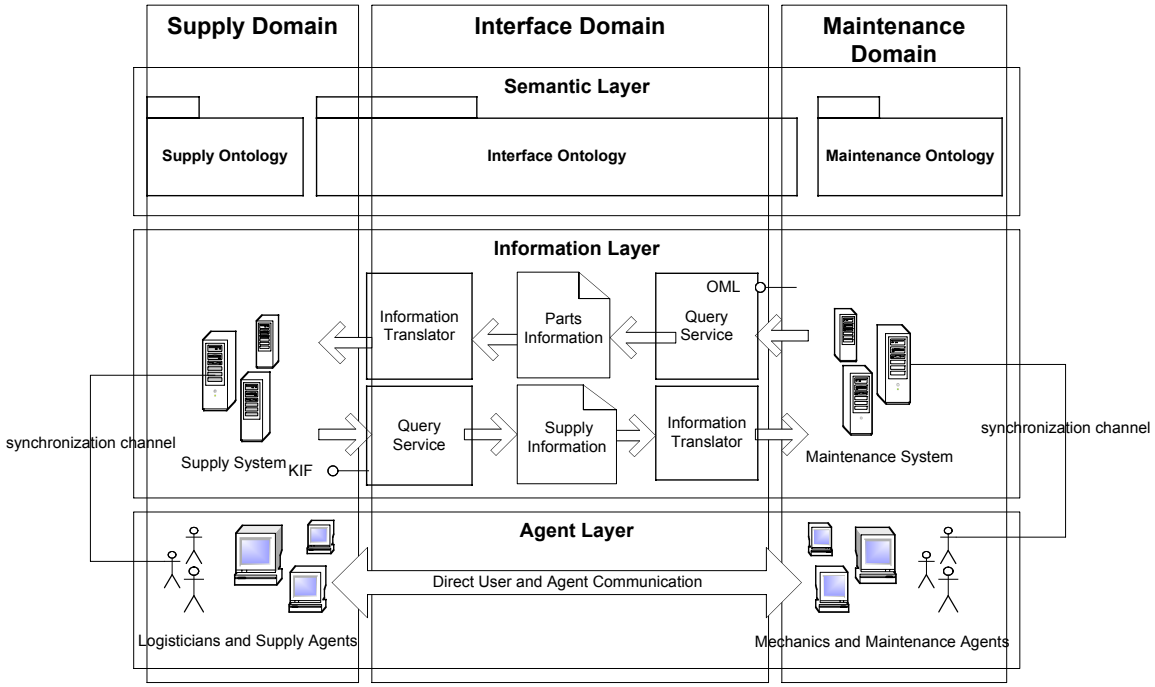


Fig.37: Information Level System Interface

The interface ontology itself will now consist of multiple interrelated entities derived from a Upper Level Interface Domain Ontology that provides higher level semantic context to each entity type concretely defined and used in the interface proper. The Interface Ontology should also define the entities required to pull the interface information from the context of one system and to place it into the context of another. Although these constructs may not be present in the physical implementations that transport the information from one system to another it is important that they are defined in the ontology to fully capture the semantic context of the information. The Upper Level Interface Domain Ontology may have existed prior to the development of the interface or may have been developed in conjunction with it. In either case it is important that the application level ontologies specific to the individual Supply and Maintenance Domains in turn utilize it directly or at least reference it by semantically marking up the entities in the ontologies of these domains to correlate them to the concepts defined by the Upper Level Interface Domain Ontology.

With a semantic layer thus defined the information level interface can do much more than generate simple fixed reports. Each system can expose a much more generalized query interface. The queries are formulated and the responses returned in terms of the entities defined in the interface domain ontology. This allows for a much more flexible interface that is more likely to survive evolving interface and system requirements over time. Note that in order to support a generalized query interface at least one additional interface ontology must be defined that defines the semantics of the queries, or commands that in turn uses the interface domain ontology as logical arguments. For this purpose, many well defined standards exist such as Structured Query Language (SQL) and Knowledge Interchange Format (KIF), or systems may expose their own proprietary but publicly defined interface such as the ICDM-based Object Management Layer (OML) employed by most of the systems developed by CDM Technologies.

Between the information and agent layers in Fig.37 are depicted synchronization channels. In this example the Maintenance and Supply systems are information-centric systems that provide

for the development of software agents by providing subscription services to client applications. A subscription service is key to agent development as it allows an agent to register for the ontological patterns that trigger it to action. In this way, agents are able to operate in support of their users, as they are always ready to act in fulfillment of their responsibilities without having to perform needless work querying the information store for conditions that may never arise.

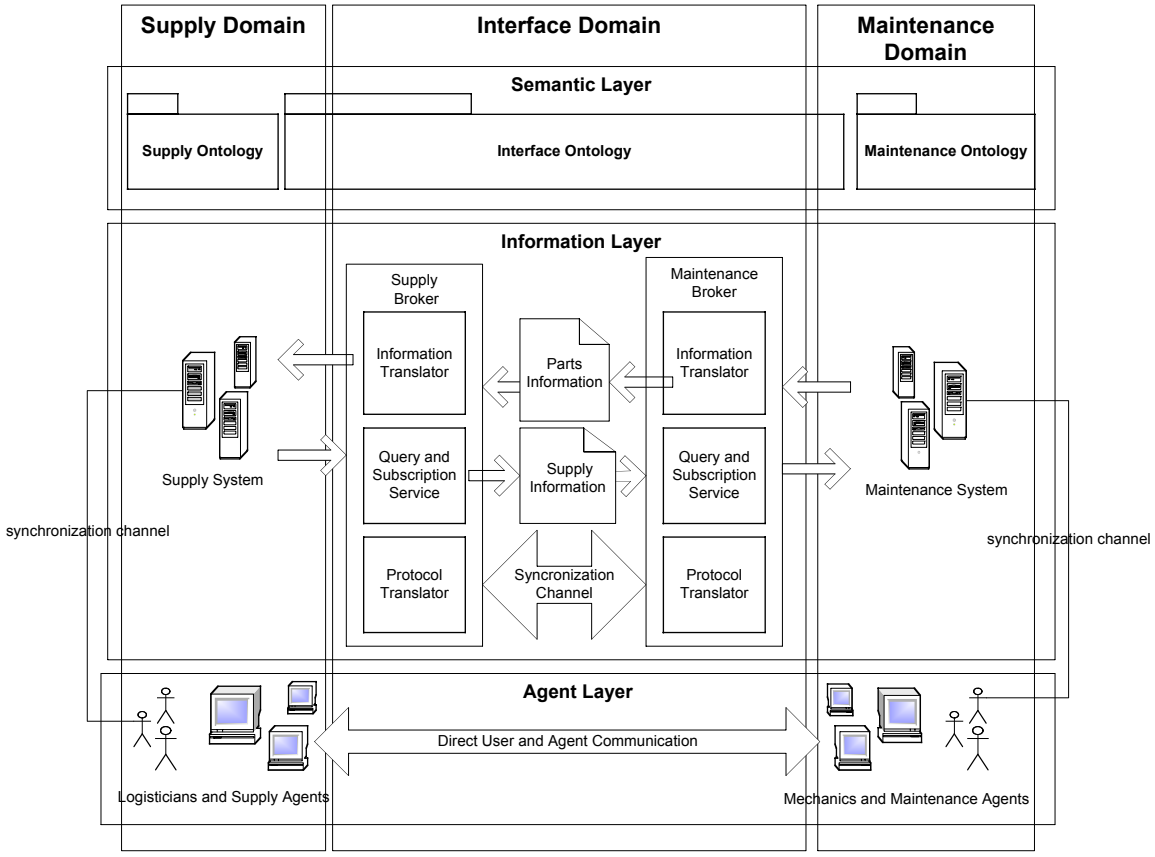


Fig.38: Information level system interoperability

6.1.4 Information Level Interoperability

The Information Level Interface of the previous section has a shortcoming in that the Supply System and Maintenance system must both explicitly query each other to receive new information. Whether or not any new information is available a query must still be run just to find out. On the flip side, immediately after a query has been run information could change in the source system that would not be reflected in the querying system until after processing the next query, which may take a while depending on the polling schedule employed. This situation can be remedied by employing the same sort of synchronization channel used between the individual information-centric systems and the agents they support that is show in Fig.38. The addition of a synchronization channel for the interface allows for the development of interface brokers. Interface brokers serve as agents in the systems they support by automatically synchronizing the state of the system to the state of interest in an external system via the defined interface between the systems.

This approach allows for true interoperability between the systems but is not without its own difficulties. Many of the entities exchanged between the systems correlate to items in the real

world and thus have unique identities whose keys must be managed within the confines of a real system implementation. In this sort of information level interface this is typically accomplished by designating a single specific source for each type of unique entity. While this approach works well for interfaces in which only a few systems are participating it starts to break down in larger interoperability scenarios as each system broker must know about all the other systems participating and which system is designated to be the definitive source of which data. This approach requires much duplication of effort within the individual data brokers and introduces an undesirable coupling between the systems. One approach for dealing with this is the introduction of an interoperability server.

6.1.5 Interoperability Server

An interoperability server elevates the interfaces between systems to the level of information-centric systems themselves. This approach provides one common implementation of the individual system data brokers that know in which system or combination of systems to find information defined within the Interface Domain Ontology. It also provides specifically for the management of unique entities that are shared across two or more of the interfacing systems.

Employing the concept of an interoperability server leads to a system of systems architecture that groups collections of systems that need to regularly exchange information into loosely coupled federations whose central hub consists of a specific instance of an agent-based interoperability server that is configured to address the specific needs of the federation. This concept views the interoperability server as just another system that allows one to layer a hierarchy on this system of systems architecture where higher level federations may include as systems zero or more interoperability servers typically from lower level federations. Within the defense domain, one could envision the proposed system of systems hierarchy following along the lines of existing unit hierarchies within the individual services with the top level of the hierarchy operating at the level of the Joint Chiefs of Staff, and Commander in Chief. Of course, cross-ties between the individual units and services at the lower levels of the hierarchy may also exist. The end-state of this vision is a single, albeit large and distributed, system of systems that incorporates the entire information infrastructure of an enterprise. This system of systems is tailored to meet the specific needs of user communities at all levels, by utilizing the systems specifically developed to meet their local needs, and is adaptable to change due to the loose coupling between systems.

There will be several distinct ontologies associated with each Interoperability Server (Fig.39). System Interface Ontologies that are unique to each system participating in the federation will be used to define the interrelated logical constructs within the corresponding system that are targeted to participate in external interactions. For example, the System Interface Ontology for an air load planning system may include constructs to represent air transports, stow areas, and cargo items. Also associated with each participating system is an ontological map that defines the transformations required to translate information represented in the corresponding System Interface Ontology both to and from the Federation Interface Ontology. Federation Interface Ontologies that are unique to each federation will be used to define the interrelated logical constructs with which the client systems to the Interoperability Server may interact. This ontology defines all the information of common interest to the entire federation as opposed to the specialized interests of the individual systems that are participating in it.

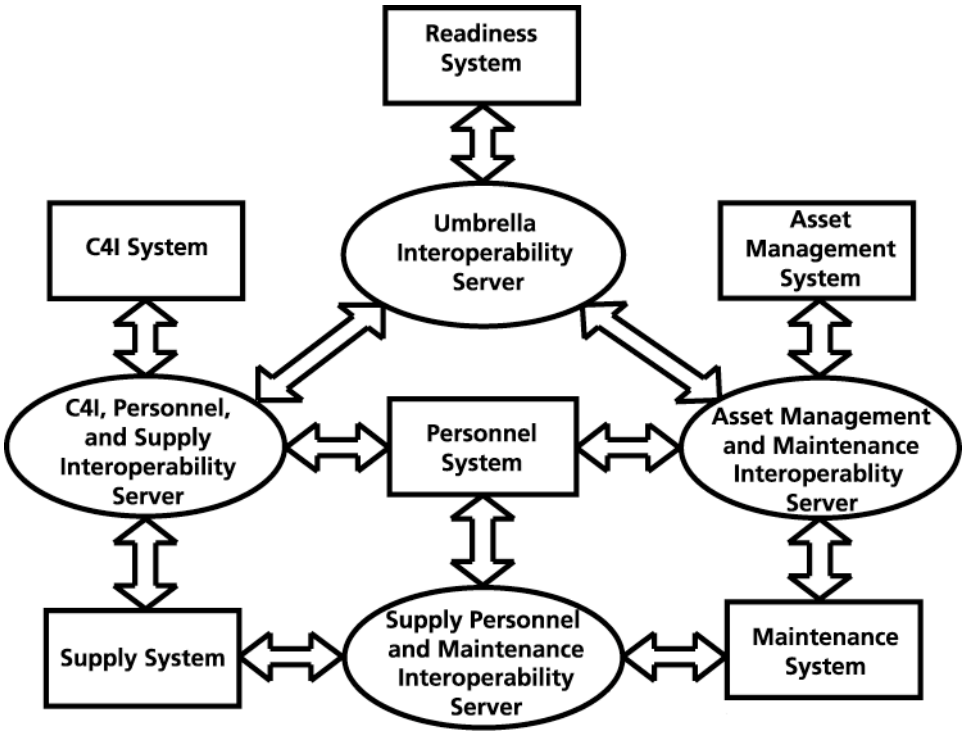


Fig.39: Interoperability Server

For example, the Federation Interface Ontology for a joint logistics transport federation, which interfaces specialized air, rail, and sea load planning systems may define a transport construct which is a generalization of the specialized air transport, rail transport, and sea transport constructs that may be defined by individual System Interface Ontologies of the participating systems. This ontology once established serves as a standard for the domain represented by the federation similar in concept to the enterprise models that were popularized in the late 1980s and early 1990s such as the DoD Logical Data Model but exist within a more manageable scope and are driven by the interoperability requirements of the federation. Finally, an Interoperability Ontology shared by all Interoperability Servers can be used to define the interrelated logical constructs associated with interoperating systems and the services provided by the interoperability server. These constructs are independent of the logical domain entities associated with a specific federation. For example, the Interoperability Ontology may define constructs such as query, constraint, system, and ontology.

The key to the interoperability between systems lies with well-defined system and interface ontologies. An ontology makes explicit the conceptualizations used and shared by the interoperating systems. The shared conceptualization is known as the interface domain ontology. The interface domain ontology and the individual system domain ontologies should both be well marked up in terms of each other and ideally share both an upper level and lower level interface domain ontology. In this way, the mappings that determine the context of interfaced entities on either side of the exchange are made explicit and are more likely to endure evolutionary changes to the systems and local modifications or special case usages. For systems to truly interoperate, rather than just interface some sort of synchronization channel must be provided. As the number of systems participating in an interface grows even a well-designed information level interface can become unmanageable and an interoperability server approach should be considered.

6.2 The Concept of Semantic Filters

As discussed in the previous section many of the obstacles to interoperability and integration are largely overcome in an information-centric software systems environment by representing adequate context to support automated reasoning capabilities. An integral part of such a capability is the inclusion of a subscription service that allows the interests to be described in the explicit terms of the information model(s). The technology that is most commonly employed to achieve this level of representation and 'understanding' in a software application is an ***ontology***.

An ontology in this sense can be defined as a constraint, abstraction and relationship rich model (or set of models) describing the entities, concepts, and notions relevant to the domain of operations. The problem arises when two or more of these systems, each operating over a potentially extensive set of descriptions attempt to collaborate with each other. While collaboration within each of these systems may be based on very high-level descriptions, it will undoubtedly be subject to various application-specific biases. For example, in a tactical command and control system an entity such as a M1A1 tank may be viewed, and therefore represented as a tactical asset. In this case the bias would be toward tactical utility. However, in a logistics system the same M1A1 tank would most appropriately be viewed as a potential supply item with emphasis on logistical inventory and supply. In both cases, however, the subject is still the exact same M1A1 tank with its basic inherent characteristics. The difference resides in the manner in which the tank is being viewed by each of these systems. Another term for this bias-based filter is ***perspective***.

Perspective is not only a natural component of the way in which we perceive the world but moreover should be viewed as a highly beneficial and desirable characteristic. Perspective is the ingredient in an ontology-based decision-support system that allows for the accurate representation of domain-specific notions and bias. For example, if a decision-support system is to assist in the formulation of logistical supply missions then it is more appropriate and beneficial for an entity such as a howitzer to be primarily viewed as a supply item instead of a tactical asset. If viewed as a supply item the description of the howitzer could provide significant detail in terms of the item's shipping weight, shipping dimensions, tie-down points, and so on, all of which is pertinent information for the transportation of the howitzer. In the context of a tactical command and control system, however, such information is of much less (if any) interest. Information that would be relevant in such a tactical system would be tactical characteristics such as projectile range, effective casualty radius, advancement velocity, etc. Conversely, these characteristics are of little or no significance in the domain of logistics. Nevertheless, regardless of the perspective it may be the exact same howitzer that is being discussed between the two disparate systems (i.e., logistics system and tactical system). However, it is being discussed within two different contexts exhibiting two distinctly different perspectives. While collaboration within or across systems supported by the exact same perspective-based representation performs well, the problem arises when collaboration needs to occur between systems or system components where the perspectives are in fact not the same and potentially drastically dissimilar. In this unto common case, the extent to which systems can effectively collaborate on events and information, without a means of translation, is essentially limited to low-level data passing with receivers having little or no understanding of content and implication. Simply stated, the problem at the heart of interoperability between decision-support systems resides in the means by which information-centric systems exhibiting wholly, or even partially disparate perspectives, can interoperate at a meaningful and useful level.

The solution to this dilemma can essentially take two different directions. The first of these paths focuses on the development of a 'universal' ontology. Such an ontology would represent a single view of the world covering all relevant domains. Each system would utilize this representation as the core informational basis for operation. Since each system would have knowledge of this common representation of the entities, notions, and concepts, interoperability at the information level would be clear and concise requiring no potentially context-diminishing translation. However, as straightforward as this may appear there are two major flaws with this approach. First, in practicality it is highly unlikely that such a universal description could actually be successfully developed. Considering the amount of forethought and vision this task would require, such an undertaking would be of monumental scale as well as being plagued with misrepresentation. Inevitably, certain notions or concepts would be inappropriately represented in a particular domain in an effort to model them adequately in another.

The second flaw with the universal ontology approach is less obvious but perhaps even more limiting. Considering the number of domains across which such an all-encompassing ontology would need to extend, the resulting ontology would most likely be comprised mainly of generalities. While useful for some types of analyses these generalities would typically only partially represent the manner in which any one particular system wished to *see the world*. In other words, due to the number of perspectives a universal ontology would attempt to represent, the resulting ontology would ironically end up being just the opposite, a perspective-absent description essentially devoid of any domain-specific detail and falling far short of system needs and expectations. While perspective was the cause of the original interoperability problem it is still a highly valuable characteristic that should not only be preserved but should be wholeheartedly embraced and promoted. As mentioned earlier, perspective is a valuable and useful means of conveying domain-specific notions and bias, which are crucial to information-centric decision-support systems. To omit its presence is to significantly reduce the usefulness of an ontology and therefore the effectiveness of the utilizing decision-support system(s). This coupled with the highly unlikely potential for developing such a comprehensive, inter-domain description of the world renders the universal ontology approach both unrealistic and wholly ineffective.

The second, more promising solution to interoperability between decision-support systems introduces the notion of a ***perspective filter***. Based on the façade design pattern (Buschmann et al. 1996, Fowler 1997) perspective filters allow core entities, concepts and notions accessible to interoperating systems to be viewed in a more appropriate form relative to each collaborator's perspective. In brief, the façade pattern allows for a certain description to be viewed, and consequently interacted with in a more native manner. Similar to a pair of infrared *night vision* goggles, overlaying a filter may enhance or refine otherwise limited information. In the case of ontology-based collaboration this filter essentially superimposes a more perspective-oriented, ontological layer over the initial representation. The filter may not only add or modify the terminology and constraints of the core descriptions but may also extend and enhance it through the incorporation of additional characteristics. These characteristics may take the form of additional attributes and relationships as well as refining existing constraints, or even adding entirely new constraints. For example, Fig.40 illustrates the use of a logistically oriented perspective filter over a core description of conveyances. Note first that while the core conveyance ontology appears to represent only a limited amount of bias the effectiveness of perspective filters certainly does not require such a general core description. If the core ontology were heavily biased toward a foreign set of perspectives it would simply mean that the perspective filters would need to be more extensive and incorporate additional constraints,

extensions, etc. However, for clarity of illustration a limited, rather general core ontology has been selected.

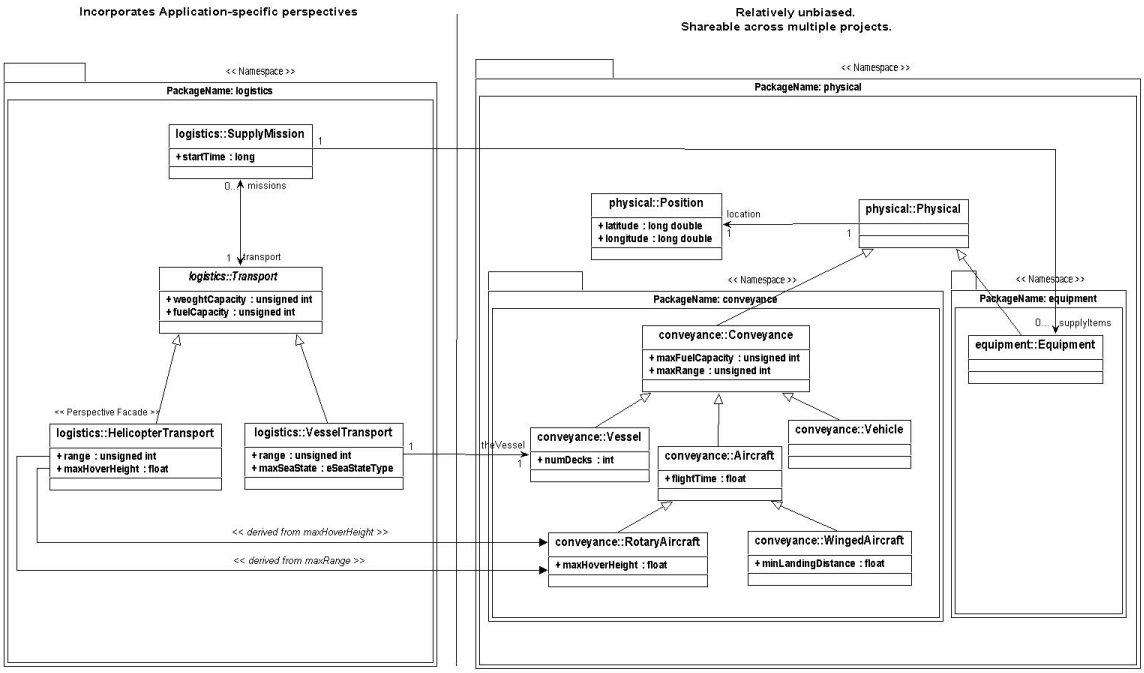


Fig.40: Partially derived logistics ontology

Core of the logistics perspective presented in Fig.40 is the notion of a transport. However, although the logistics system may have a notion of all of the types of conveyances (i.e., vessels, vehicles, and aircraft) represented in the core ontology, in the context of this example, it may only consider vessels and rotary aircraft as potential transports. In this situation it would be valuable to represent this refined constraint in the ontology forming the representational heart of the logistics system while still employing the core conveyance ontology. As Fig.40 illustrates, representing such refinement can be accomplished by explicitly introducing a constrained notion of a transport in the application-specific filter ontology. An abstract *Transport* is defined to have two specific derivations (*VesselTransport* and *HelicopterTransport*). At this point it is immediately apparent that a vehicle is not a transport candidate. In the context of the example, logistics system transports can only be *VesselTransports* or *HelicopterTransports*. The task now becomes linking these two system specific notions to the core conveyance ontology.

Relating these two transport types to their conveyance ontology counterparts can be achieved in two different ways. For illustration purposes, the definition of *VesselTransport* adopts the first method while *HelicopterTransport* employs the second. The first method defines an explicit relationship between the *VesselTransport* and the core description of a vessel outlined in the conveyance ontology. Utilizing this approach, obtaining the core information relative to the corresponding *Vessel* from a *VesselTransport* requires both knowledge of their relationship in addition to another level of indirection. For reasons of performance and representational accuracy, both of these requirements may be undesirable.

The second method, illustrated in Fig.40 using *HelicopterTransport*, avoids both shortcomings inherent in the first approach. In this case, *HelicopterTransport* exists as a façade, or filter, which transparently links at the attribute level into the core *RotaryAircraft* description. That is, each attribute of *RotaryAircraft* desired to be exposed to users of *HelicopterTransport* is explicitly declared in the façade. For example, since the maximum range of travel is relevant to the definition of a *HelicopterTransport* the *maxRange* attribute of *RotaryAircraft* (inherited from *Conveyance*) is subsequently exposed in the HelicopterTransport façade description. By virtue of being declared in a façade any access to such an attribute would be transparently mapped into the corresponding attribute(s) on which it is based. In the case of the *range* attribute of *HelicopterTransport*, access would transparently be directed to the inherited *maxRange* attribute of *RotaryAircraft*. Notice also the use of alternative terminology over that used in the core ontology (i.e., *range* vs. *maxRange*). It should also be noted that the derivative nature of a façade attribute is not limited to mapping into another attribute. Rather, the value of a façade attribute may also be derived through calculation, perhaps based on the values of multiple attributes residing in potentially several different core objects. In either case, the fact that the value of the façade attribute is derived is completely transparent to the façade user.

Another perspective-oriented enhancement to the core ontology illustrated in Fig.40 is the notion of a *SupplyMission*. Being a fundamental concept in the example logistics system a supply mission essentially relates supply items in the form of equipment to the transports by which they will be delivered. Once again, the definition of a logistics-specific notion (i.e., supply items) is derived from a notion defined in the core ontology (i.e., equipment). In this case, an explicit relationship is declared linking *SupplyMission* to zero or more *Equipment* items. Since, from the perspective of the logistics system *Equipment* scheduled for delivery are viewed as items that are to be supplied, the term *supplyItems* is used as the referencing nomenclature. Such an enhancement demonstrates the ability to integrate new concepts (i.e., supply missions) with existing core notions.

Considerable advantages accrue from drawing relevant concepts and notions into a system's local set of perspective-rich, filter ontologies. As the above example illustrates, key components of these perspective-oriented ontologies could be derived from a set of core, relatively unbiased common notions forming the basis for informational collaboration among systems. There are several benefits to adopting this approach. Collaboration among information-centric, decision-support systems would take place in terms of various core ontologies (i.e., *Conveyance*) with each collaborating application viewing these core entities, concepts and notions according to its own perspective. Fig.41 briefly extends the logistics example presented in Fig.40 showing collaboration between the original logistics system and a tactical command and control system. Collaboration between these two example systems is in terms of the common, core ontologies on which they share their derivations. A conveyance is still a conveyance whether it is viewed in the context of logistics or tactical command and control. To represent domain-specific notions (e.g., transport, supply item, tactical asset, etc.) each collaborating system would apply the appropriate filter. Although discussing a conveyance from partially disparate perspectives both systems can collaborate about core entities, concepts, and notions.

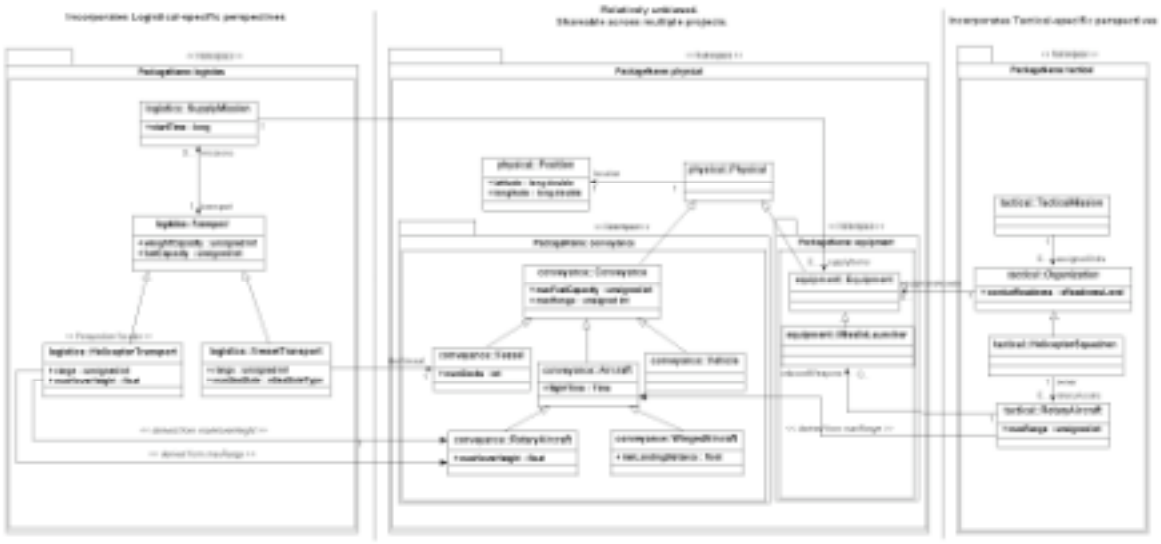


Fig.41: Two disparate domains linked into the same core ontology

Another advantage of supplementing core, non-system-specific ontologies with perspective rich filters is the preservation of both time and effort during the development of such information-centric systems. Core ontologies could be archived in an ontology library forming a useful reference source for the development of new system ontologies. Models created for new decision-support systems could make use of this ontology library as a strong basis for deriving system-specific filters. In addition, such a process would promote the use of common core descriptions increasing the potential for interoperability and component reuse even further.

6.3 Use-Case Centered Development Process

The principal purpose of any software application is to provide value to end-users. Several questions then arise: Who are the end-users? What decisions do they need to make? How would access to integrated data and information (i.e., data in context) help in the decision process? In general, these are all aspects of a single question: If users had access to all of the data in all of the systems used anywhere within their operational environment, what would they want to do with these data?

These questions are complicated by the fact that a software applications environment is bound to change the way in which users perform their work, and will certainly create the possibility that they will be doing different kinds of work than they do now, once a particular application is in operation. As a result, it is impossible to determine what the real requirements of the application or system will be once it is built, because no one can precisely predict what kinds of changes are likely to take place.

The solution lies in an ***iterative development*** process. The guiding principle of iterative development is to deliver functional software at short intervals to end-users who then provide feedback and guidance on future requirements. The process of defining requirements becomes incremental, and the basis of collaboration between end-users and system designers. Since end-users know how they perform their work under current conditions, they must be considered an important source of input for defining implementation priorities. While designers and developers

can foresee future possibilities, they typically cannot predict whether a given piece of functionality will in fact become an important capability for the end-users. Both have knowledge that can guide development, and both are necessary for a successful system. In order for this concept to be realized, it is important to stay focused on the needs of the users. This approach is often referred to as ***use-case centered*** development.

There are many forms of use-cases, differing primarily in their relative formality (Cockburn 2001). Basically, a use-case is a story that tells how an 'actor' will interact with the 'system'. Actors can be either human users or other systems. The 'system' can be either the entire system or just a part of a system, depending on the objectives and role of the 'actor' for a given use-case. Use-cases can provide the basis for requirements discovery and definition. As such, they describe the actor's view of the system under discussion. Use-cases describe the behavior of the system given input from the actor, but only that behavior that the actor is aware of (plus important side effects, if any). However, use-cases do not include details of system implementation or internal design such as data models. They also do not describe the user interface.

A complete use-case includes alternate paths (referred to as 'extensions'), which describe all the situations under which either the actor or the system can perform different actions based on the current state. The use-case also includes failure scenarios (i.e., conditions under which the system is not able to support the user's goal), along with pre-conditions (i.e., what must be true before the use-case can be executed) and guarantees (i.e., what will be true after the use-case has been successfully executed). Each use-case constitutes a contract for the behavior of the system. To facilitate the implementation of an ICDM-based application, it would be very helpful to draw up use-cases that describe how the potential users currently accomplish their goals. Knowledge of current planning and decision-making processes will provide invaluable information to software developers and program managers to determine the data sources and information models (i.e., ontologies) that will be needed in order to implement these existing use-cases in the new knowledge management environment.

6.3.1 Use-Cases and Iterative Development

A set of use-cases can form the starting point for a development process. In an iterative development process, the system is implemented incrementally and delivered to end-users as soon and as often as possible. As users receive successive versions of the software, their responses frequently result in new or modified use-cases, which must be incorporated in future iterations. New requirements are discovered as users and developers work with the system.

This is in contrast to processes that attempt to specify all the requirements before development begins. Comparatively, iterative development processes tend to produce systems that are more accepted by users, since developers are able to respond to changing goals and needs as implementation progresses. This characteristic is especially important for ontology-based systems, which are likely and intended to change the way that people perform their work. Requirements for such systems will evolve as users see new possibilities.

For the implementation of the Information Layer of a multi-layered knowledge management system environment, use-case oriented iterative development should begin by identifying major stakeholders and user categories. For each use category, it is important to find a representative user to provide input and perspective. The initial impetus for building any existing corporate data environment was undoubtedly at least partly driven by a desire to reduce, and if possible eliminate, the need for planning staff and decision makers to manually bring together data from multiple existing systems in order to accomplish their goals. If this is the case, it is vital to

include representatives of these planners and decision makers in the initial group of stakeholders and users.

The process of discovering use-cases will begin by listing examples of situations requiring multiple data sources. Each example should include the reason for bringing together these data, the list of data sources, the method of data extraction (e.g., existing client applications, direct database queries, etc.) and the type of data retrieved from each source. From this information, an ***as is*** use-case can be defined, followed by the corresponding ***to be*** use-case, which describes the user's interaction with the planned system.

The initial set of use-cases should be prioritized to ensure that the most important interactions are implemented early in the project's lifetime. The criteria for use-case prioritization are primarily user-oriented (e.g., How often does the user need to execute this use-case? How much time does it take to gather the data? How significant is the result likely to be?). However, especially during the early stages of the development cycle, developer priorities are also critical. Use-cases may require building significant parts of the planned system architecture, or they may involve parts of the architecture that developers see as risky. Both of these situations would cause a use-case to assume a higher priority from a developer's point of view, since the architecture should be built as soon as possible and potential risks should be addressed early in the project when alternatives are still available should the risk prove insurmountable.

Priorities may also be affected by the data sources involved in a use-case. To the extent possible, each implemented use-case should include one or more data sources that have already been integrated into the system in earlier use-cases, and in addition should include a data source that is not yet part of the proposed new information-centric (i.e., ontology-based) system environment. In this way, successive development iterations will build on previous work, while gradually extending the range of integration.

Once the first set of use-cases has been prioritized, developers will determine how many use-cases can be reasonably implemented in the first development cycle. Due to the complexity of integrating new data sources into a new information-centric environment, it is likely that the first cycle will be somewhat lengthy; – possibly as much as six months long. This duration must be estimated in large part by the data source integration team, based on the specific data sources involved. The first cycle is likely to consist almost entirely of building integration and architectural infrastructure, but will also include a (possibly small) number of use-cases. The key goal of all iterative development processes is that at the end of each development cycle, there will be releasable software. Whether a particular version of the system is released for use or not is a decision that is likely to be made outside the development team, but each development cycle should result in a system that can be released if that decision is made. If at all possible the product of the first cycle should be released at least to the user groups whose use-cases are included in the system.

Later development cycles will follow essentially the same pattern. As users work with the evolving system, they will generate new use-cases and extensions to existing use-cases. Some older use-cases may become obsolete as the new system changes the way that users are working. Also, it is likely that developers will see ways in which some use-cases can be modified by constructing agents to determine that a user may benefit from specific information. And, as more data sources are added to the Operational Data Store or Data Warehouse, new types of processing (e.g., OLAP and Data Mining applications) may become both possible and useful. This, in turn, will also increase the number of potential use-cases.

At the beginning of each development cycle, new and old use-cases should be prioritized together, and developers will again determine which ones can be attempted for the next release. Developer priorities for subsequent cycles will include looking for use-cases that allow them to extend functionality created for prior cycles. After the first cycle, the time between releases can be shortened so that users can see the system evolving quickly. This increases the chances of user acceptance, since the effects of their requests for change can be seen over a relatively short period of time.

As the functionality of the system progressively increases, new user groups can be included. These might include, among others, the users of some of the systems that will feed information into the new system environment. These users might benefit from access to a wider range of data and information than is provided by the system they are currently using. There may also be opportunities to simplify or enhance their work, through the use of information layer capabilities on top of the same data that they currently use.

Iterative use-case centered development processes tend to produce software systems that are accepted by end-users, for several reasons. First, the end-users themselves are directly involved in defining requirements. Second, end-users see the system at an early stage and as it evolves. At each release, users have an opportunity to correct the direction that the development team is moving, and to add new requirements. Third, the requirements implemented during each development cycle are the highest priority, based on the input of all stakeholders, including the users themselves. Together, these aspects of iterative development ensure that at any point in time, the system meets the most important user needs.

7. References and Bibliography

7.1 References

Ayers D., H. Bergsten, M. Bogovich, J. Diamond, M. Ferris, M. Fleury, A. Halberstadt, P. Houle, P. Mohseni, A. Patzer, R. Phillips, S. Li, K. Vedati, M. Wilcox, and S. Zeiger (1999); 'Professional Java Server Programming'; Wrox Press, Birmingham, UK.

Bancilhon F., C. Delobel and P. Kanellakis (eds.) (1992); 'Building an Object-Oriented Database Systems'; Morgan Kaufmann, San Mateo, California.

Buschmann F, D Schmidt, H Rohnert, and M Stal (1996); 'Pattern-Oriented Software Architecture: A System of Patterns'; Vols. 1 and 2, John Wiley and Sons, New York, New York.

Chandrasekaran, B., Josephson, John R. and Benjamins, V. Richard. What are Ontologies, and Why Do We Need Them? IEEE Intelligent Systems, Vol. 14 No. 1, January/February 1999.

Cockburn A. (2001); 'Writing Effective Use Cases'; Addison-Wesley, Reading, Massachusetts.

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

Forgy C. (1982); 'Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem'; Artificial Intelligence, Vol.19 (pp.17-37).

Fowler M. and K. Scott (1997); 'UML Distilled: Applying the Standard Object Modeling Language'; Addison-Wesley, Reading, Massachusetts.

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

Gray S. and R. Lievano (1997); 'Microsoft Transaction Server 2.0'; SAMS Publishing, Indianapolis, Indiana.

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.

IONA (1996); 'Orbix Web: Programming Guide'; IONA Technologies Ltd., Dublin, Ireland.

Lewis B. and D. Berg (1996); 'Threads Primer: A Guide to Multithreaded Programming'; SunSoft Press, Mountain View, California.

Minsky M. (1990); 'Logical vs. Analogical or Symbolic vs. Connectionist or Neat vs. Scruffy'; in Winston and Shellard (eds.) Artificial Intelligence at MIT, vol.1, MIT Press, Cambridge, Massachusetts, (pp. 218-243).

Mowbray T. and R. Zahavi; 'The Essential CORBA: Systems Integration Using Distributed Objects'; Wiley, New York, New York, 1995.

NASA (1992); 'CLIPS 6.0 Reference Manual'; Software Technologies Branch, Lyndon B. Johnson Space Center, Houston, Texas.

Orfali R., D. Harkey and J. Edwards (1996); 'The Essential Distributed Objects Survival Guide'; John Wiley and Sons, Inc., New York, New York.

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

Pohl J., A. Chapman and K. Pohl (2000); 'Computer-Aided Design Systems for the 21st Century: Some Design Guidelines'; 5th International Conference on Design and Decision-Support Systems for Architecture and Urban Planning, Nijkerk, The Netherlands, August 22-25.

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. Wood, K.J. Pohl and A. Chapman (1999); 'IMMACCS: A Military Decision-Support System'; Proc. DARPA-JFACC Symposium on Advances in Enterprise Control, San Diego, California, Nov.15-16.

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. (1997); 'Human-Computer Partnership in Decision-Support Systems: Some Design Guidelines'; in Pohl J. (ed.) Advances in Collaborative Design and Decision-Support Systems, focus symposium: International Conference on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, August 18-22 (pp.71-82).

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 J., L. Myers, A. Chapman, J. Snyder, H. Chauvet, J. Cotton, C. Johnson and D. Johnson (1991); 'ICADS Working Model Version 2 and Future Directions'; Technical Report, CADRU-05-91, CAD Research Center, Design Institute, College of Architecture and Environmental Design, Cal Poly, San Luis Obispo, CA.

Pohl K. (2001); 'Perspective Filters as a Means for Interoperability Among Information-Centric Decision-Support Systems'; Office of Naval Research (ONR) Workshop hosted by the CAD Research Center in Quantico, VA, June 5-7.

Pohl K. (1997); 'ICDM: A Design and Execution Toolkit for Agent-Based, Decision-Support Applications'; in Pohl J. (ed.) Advances in Collaborative Design and Decision-Support Systems, focus symposium: International Conference on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, August 18-22 (pp.101-110).

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.

Pohl J., L. Myers, J. Cotton, A. Chapman, J. Snyder, H. Chauvet, K. Pohl and J. La Porta; 'A Computer-Based Design Environment: Implemented and Planned Extensions of the ICADS Model'; Technical Report, CADRU-06-92, CAD Research Center, Design and Construction Institute, College of Architecture and Environmental Design, Cal Poly, San Luis Obispo, CA, 1992.

Uschold, Mike and Gruninger, Michael. Ontologies: Principles, Methods and Applications. Knowledge Engineering Review, Vol. 11, No. 2, June 1996.

Uschold, Mike and King, Martin. Towards a Methodology for Building Ontologies. Workshop on 'Basic Ontological Issues in Knowledge Sharing' held in conjunction with IJCAI-95.

7.2 Bibliography

Knowledge Management

O'Dell C., S. Elliott, and C. Hubert (2000); 'Knowledge Management: A Guide for Your Journey to Best-Practice Processes'; APQC's Passport to Success Series, American Productivity and Quality Center, Houston, Texas.

O'Dell C., F. Hasanali, C. Hubert, K. Lopez, and C. Raybourn (2000); 'Stages of Implementation: A Guide for Your Journey to Best-Practice Processes'; APQC's Passport to Success Series, American Productivity and Quality Center, Houston, Texas.

O'Leary D. and P. Selfridge (1999); 'Knowledge Management for Best Practices'; Intelligence, Winter (pp.12-23).

Pohl J. (2001); 'Transition from Data to Information'; InterSymp-2001, 13th International Conference on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, July 30 – August 3, 2001.

Pohl J. (2003); 'The Emerging Management Paradigm: Some Organizational and Technical Issues; InterSymp-2003, 15th International Conference on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, July 29 – August 1, 2003.

Smith G. and A. Farquhar (2000); 'The Road Ahead for Knowledge Management'; AI Magazine, Winter 2000 (pp.17-40).

Interoperability

Chaudri V., A. Farquhar, R. Fikes, P. Karp, and J. Rice (1998); 'OKBC: A Programmatic Foundation for Knowledge Base Interoperability'; Fifth National Conference on Artificial Intelligence, Menlo Park, California, AAAI Proceedings (pp.600-607).

Dawson R. (2000); 'Developing Knowledge-Based Client Relationships: The Future of Professional Services'; Butterworth-Heinemann, Boston, Massachusetts.

Dixon M. (2000); 'Common Knowledge: How Companies Thrive by Sharing What They Know'; Harvard Business School Press, Boston, Massachusetts.

Pohl J. (2001); 'Information-Centric Decision-Support Systems: A Blueprint for *Interoperability*'; Office of Naval Research, Workshop on Collaborative Decision-Support Systems, Quantico, VA, June 5-7, 2001.

Ontology-Based Software

Ambler S. (2002); 'Agile Modeling: Effective Practices for Extreme Programming and the Unified Process'; Wiley, New York, NY, 2002.

Booch G. J. Rumbaugh, and I. Jacobson (1997); 'The Unified Modeling Language (UML) User Guide'; Addison-Wesley, Reading Massachusetts.

Chandrasekaran B., J. R. Josephson, and V. R. Benjamins (1999); 'What are Ontologies, and Why Do We Need Them? IEEE Intelligent Systems, 14(1), January/February.

Cunningham W. and R. Johnson (1997); 'Analysis Patterns: Reusable Object Models'; Addison-Wesley, Reading, MA, 1997.

Farquhar A, R. Fikes and J. Rice (1997); 'The Ontolingua Server: A Tool for Collaborative Ontology Construction'; International Journal for Human-Computer Studies, 46(6), (pp.707-727).

Fowler M. (2003); 'Patterns of Enterprise Application Architecture'; Addison-Wesley, Reading, MA, 2003.

Fridman-Noy N. and C. D. Hafner (1997); 'The State of the Art in Ontology Design: A Survey and Comparative Review'; AI Magazine, 18(3), Fall.

Kruchten P. (2000); 'The Rational Unified Process: An Introduction'; Addison-Wesley, Reading, MA, 2000.

Larman C. (1998); 'Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design'; Prentice Hall, Upper Saddle River, NJ, 1998.

Leighton R. (2000); 'Information Representation Basis of Decision Support Systems'; Office of Naval Research, Workshop on Collaborative Decision-Support Systems, Embassy Suites Hotel, San Luis Obispo, CA, May 2-4, 2000.

Noy N. and C. Hafner (1997); 'The State of the Art in Ontology Design: A Survey and Comparative Review'; AI Magazine, Fall 1997 (pp.53-74).

Noy N. and D. McGuinness (2001); 'Ontology Development 101: A Guide to Creating Your First Ontology'; Stanford University, Stanford, California [www.smi.Stanford.edu]

Pohl K. (2001); 'Perspective Filters as a Means for Interoperability Among Information-Centric Decision-Support Systems'; Office of Naval Research, Workshop on Collaborative Decision-Support Systems, Quantico, VA, June 5-7, 2001.

Rumbaugh J., M. Blaha, W. Premerlani, F. Eddy and W. Lorensen (1991); 'Object-Oriented Modeling and Design'; Prentice Hall, Englewood Cliffs, NJ, 1991.

Taylor D. (1990); 'Object-Oriented Technology: A Manager's Guide'; Addison-Wesley, Reading MA, 1990.

Uschold, M. and M. Gruninger (1996); 'Ontologies: Principles, Methods and Applications'; Knowledge Engineering Review, 11(2), June.

Uschold, M. and M. King (1995); 'Towards a Methodology for Building Ontologies'; Workshop on Basic Ontological Issues in Knowledge Sharing held in conjunction with IJCAI-95.

Warmer J. and A. Kleppe (1999); 'The Object Constraint Language: Precise Modeling with UML'; Addison-Wesley, Reading, MA, 1999.

Zang M. (2003); 'Data, Information, and Knowledge in the Context of SILS'; Office of Naval Research, Workshop on Collaborative Decision-Support Systems, Quantico, VA, Sep.18-19, 2003.

Zang M. (2003); 'The Knowledge Level Approach to Intelligent Information System Design'; InterSymp-2003, 15th International Conference on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, July 29 – August 1, 2003.

8. Keyword Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z