CDM TECHNICAL REPORT: CDM-17-04

The TIRAC™ 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 Toolkit for Information Representation and Agent Collaboration (TIRAC™) software framework for the development of intelligent decision-support applications. More technical descriptions of TIRAC™ are contained in a companion CDM Technical Report (CDM-19-03) entitled: 'The TIRAC™ Development Toolkit: Technical Description'.

TIRAC™ 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 TIRAC-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 TIRAC™ toolkit include: support of an ontology-based, information-centric, distributed 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 TIRAC™ 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 TIRAC-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 TIRAC™ 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 TIRAC-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, TIRAC™ 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, TIRAC™ 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, TIRAC™ provides a development platform for achieving interoperability by formalizing a common language and compatible representation across multiple applications.

The TIRAC™ Development Toolkit: Purpose and Overview

Table of Contents

1. Background: The Information-Centric Transformation ………………………. 5
   - 1.1 The Quest for *'Interoperability'* …………………………………… 5
   - 1.2 The Meaning of *'Information-Centric'* …………………………….. 5
   - 1.3 Definitions: Data, Information, and Knowledge ……………………. 7
   - 1.4 The Data-Centric Evolution of Computer Software ………………… 9
   - 1.5 The Representation of *'Context'* in a Computer …………………….. 11
   - 1.6 The Notion of *'Intelligent Agents'* ………………………………….. 14
   - 1.7 An Information-Centric Architecture ………………………………. 15
   - 1.8 Promises of an Information-Centric Software Environment ……….. 17

2. The TIRAC™ Framework and Toolkit……………………………………….. 21
   - 2.1 Architectural Features of TIRAC™ …………………………………. 21
     - 2.1.1 Technological Underpinnings ………………………………. 21
     - 2.1.2 The TIRAC™ Framework ………………………………….. 23
     - 2.1.3 A View-Based Approach to Context Segregation ………….. 24
   - 2.2 The TIRAC™ Software development Process ……………………… 27
   - 2.3 Origin, Proprietorship, and Licensing ……………………………….. 28

3. The Underlying Design Principles …………………………………………… 29
   - 3.1 Collaboration-Intensive ……………………………………………… 29
   - 3.2 Context: Information-Centric ………………………………………... 31
   - 3.3 Extensibility and Adaptability ………………………………………. 32
   - 3.4 Multi-Tiered and Multi-Layered ……………………………………. 33
   - 3.5 The TIRAC-Based Application Environment ………………………. 34

4. An Ontology-Based Subscription Service …………………………………… 37

5. TIRAC™ in Practice: *SecureOrigins* as an Example ……………………….. 41
   - 5.1 The SecureOrigins Problem and Solution Domain …………………. 41
     - 5.1.1 The Goods Shipment Security Problem …………………….. 41
     - 5.1.2 The *Profiling* Solution Approach …………………………… 43

5.2 The TIRAC-Based *SecureOrigins* Architecture …………………….. 45

5.2.1 A Two-Layer Architecture ……………………………….. 46
5.2.2 Structure of the SecureOrigins Ontology ……………………. 47
5.2.3 Model Maintenance ………………………………………. 50
5.2.4 Artifact Generation ………………………………………. 50
5.2.5 The Object management Layer (OML) ……………………… 50
5.2.9 SecureOrigins Agent Engine Example ………………………. 51

6. Some Software Design and Development Considerations …………………… 55

6.1 Ontology Approach to Semantic Interoperability…………………… 55

6.1.1 Interoperability Example ………………………………… 56
6.1.2 Data Level System Interface …………………………….. 57
6.1.3 Information Level Interface ……………………………… 59
6.1.4 Information Level Interoperability …………………………. 61
6.1.5 Interoperability Server ……………………………………. 62

6.2 The Concept of Semantic Filters …………………………………… 64

6.3 Use-Case Centered Development Process ………………………….. 68

6.3.1 Use-Cases and Iterative Development ………………………. 69

7. References and Bibliography ………………………………………………. 73

7.1 References ……………………………………………………….. 73

7.2 Bibliography …………………………………………………….. 76

8. Keyword Index ………………………………………………………. 79

1. Background: The Information-Centric Transformation

For the past 20 years commercial corporations and government agencies have suffered under the limitations of stove-piped computer software applications that function as discrete entities within a fragmented data-processing environment. In the US military services, 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 a 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 by focusing on the representation of *data* without context 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 15 years, industry and commerce have made significant efforts to rearrange this unorganized data into purposeful data, utilizing various kinds of database management systems. However, even in this organized form, we are still dealing with data and not information.

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

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.

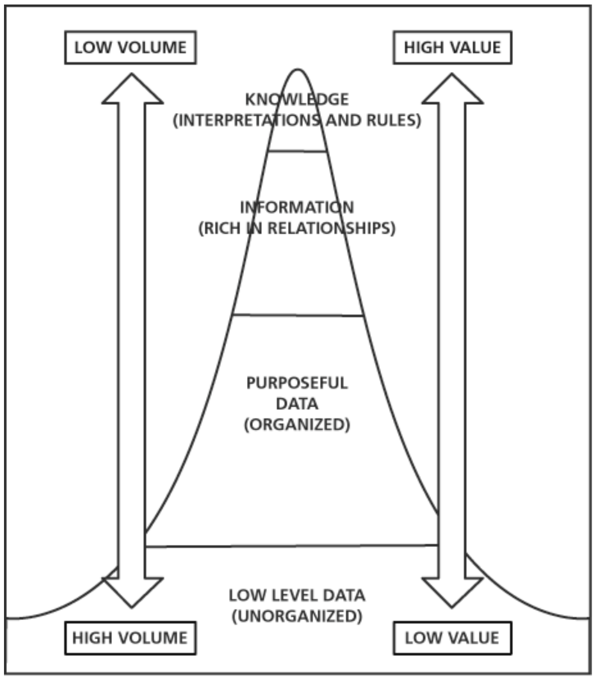

LOW VOLUME
HIGH VALUE
KNOWLEDGE (INTERPRETATIONS AND RULES)
INFORMATION (RICH IN RELATIONSHIPS)
PURPOSEFUL DATA (ORGANIZED)
LOW LEVEL DATA (UNORGANIZED)
HIGH VOLUME
LOW VALUE

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

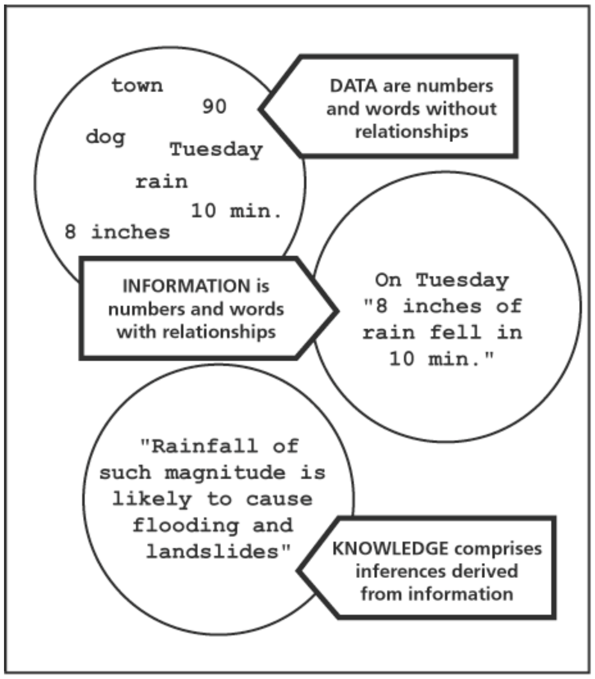

town
90
dog
Tuesday
rain
10 min.
8 inches
DATA are numbers and words without relationships
INFORMATION is numbers and words with relationships
On Tuesday "8 inches of rain fell in 10 min."
"Rainfall of such magnitude is likely to cause flooding and landslides"
KNOWLEDGE comprises inferences derived from information

Fig.2: Data, information and knowledge

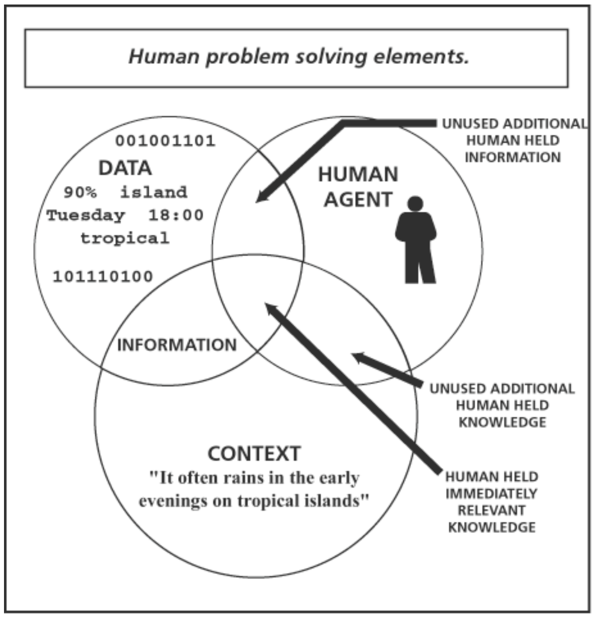

Human problem solving elements.
001001101
DATA
90% island
Tuesday 18:00
tropical
101110100
HUMAN AGENT
UNUSED ADDITIONAL HUMAN HELD INFORMATION
INFORMATION
UNUSED ADDITIONAL HUMAN HELD KNOWLEDGE
CONTEXT
"It often rains in the early evenings on tropical islands"
HUMAN HELD IMMEDIATELY RELEVANT KNOWLEDGE

Fig.3: Unassisted problem solving

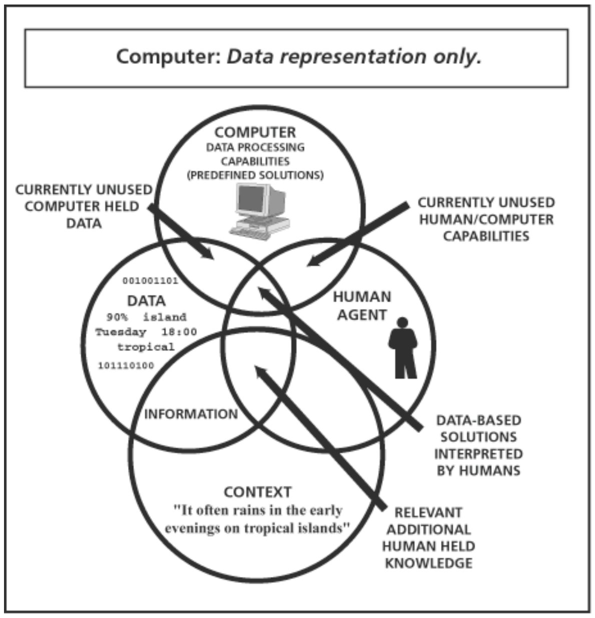

Computer: Data representation only.
COMPUTER
DATA PROCESSING CAPABILITIES (PREDEFINED SOLUTIONS)
CURRENTLY UNUSED COMPUTER HELD DATA
CURRENTLY UNUSED HUMAN/COMPUTER CAPABILITIES
001001101
DATA
90% island
Tuesday 18:00
tropical
101110100
HUMAN AGENT
INFORMATION
DATA-BASED SOLUTIONS INTERPRETED BY HUMANS
CONTEXT
"It often rains in the early evenings on tropical islands"
RELEVANT ADDITIONAL HUMAN HELD KNOWLEDGE

Fig.4: Limited data-processing assistance

1.4 The Data-Centric Evolution of Computer Software

When computers entered on the scene in the 1960s, 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.

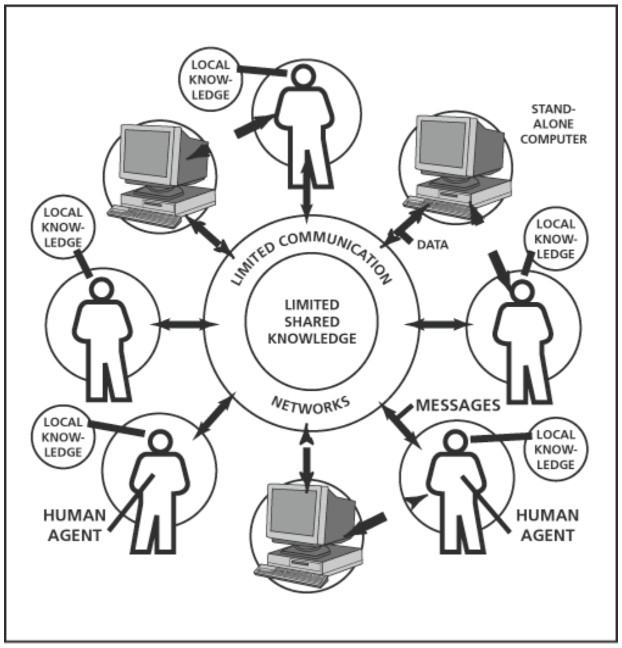


Fig.5: Limited computer assistance

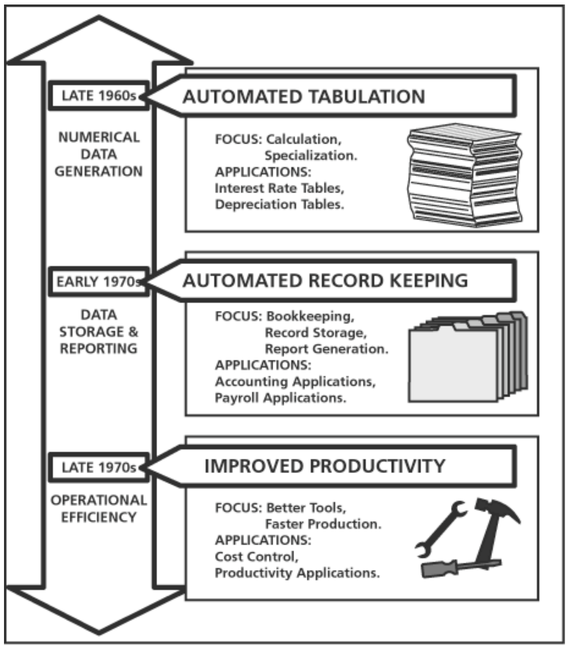
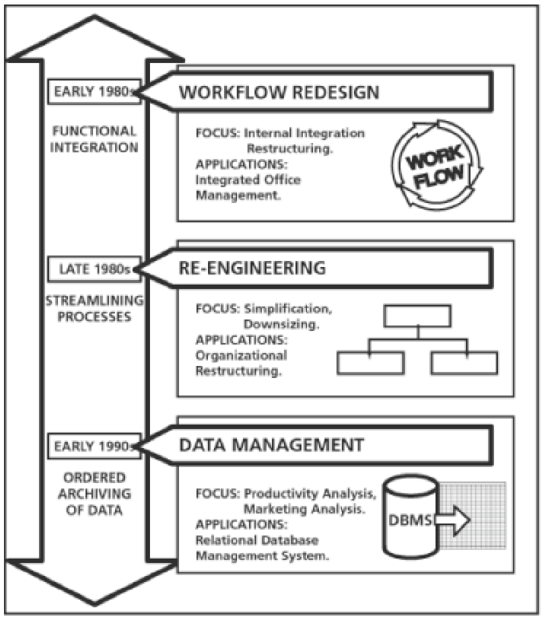

LATE 1960s
AUTOMATED TABULATION
NUMERICAL DATA GENERATION
FOCUS: Calculation, Specialization.
APPLICATIONS: Interest Rate Tables, Depreciation Tables.
EARLY 1970s
AUTOMATED RECORD KEEPING
DATA STORAGE & REPORTING
FOCUS: Bookkeeping, Record Storage, Report Generation.
APPLICATIONS: Accounting Applications, Payroll Applications.
LATE 1970s
IMPROVED PRODUCTIVITY
OPERATIONAL EFFICIENCY
FOCUS: Better Tools, Faster Production.
APPLICATIONS: Cost Control, Productivity Applications.

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 workflow as a consequence of the functional integration of the computer. More comprehensive office management applications led to the restructuring of the workflow.

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.


EARLY 1980s
FUNCTIONAL INTEGRATION
WORKFLOW REDESIGN
FOCUS: Internal Integration Restructuring.
APPLICATIONS: Integrated Office Management.
WORK FLOW
LATE 1980s
STREAMLINING PROCESSES
RE-ENGINEERING
FOCUS: Simplification, Downsizing.
APPLICATIONS: Organizational Restructuring.
EARLY 1990s
ORDERED ARCHIVING OF DATA
DATA MANAGEMENT
FOCUS: Productivity Analysis, Marketing Analysis.
APPLICATIONS: Relational Database Management System.
DBMS

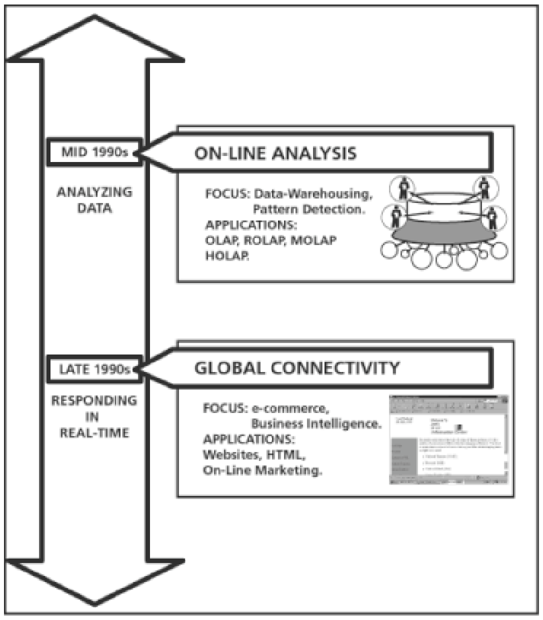

MID 1990s
ANALYZING DATA
ON-LINE ANALYSIS
FOCUS: Data-Warehousing, Pattern Detection.
APPLICATIONS: OLAP, ROLAP, MOLAP HOLAP.
LATE 1990s
RESPONDING IN REAL-TIME
GLOBAL CONNECTIVITY
FOCUS: e-commerce, Business Intelligence.
APPLICATIONS: Websites, HTML, On-Line Marketing.

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 near 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 natural language processing tools utilizing statistical and heuristic approaches (Denis 2000, Verity 1997).

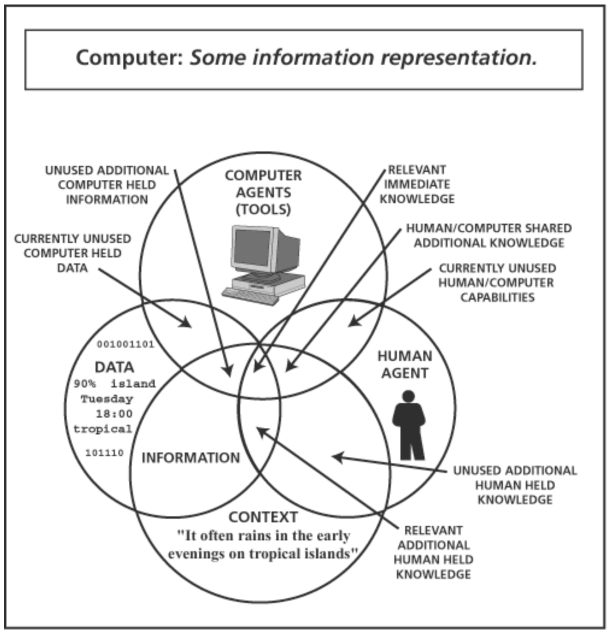
The major obstacle confronted by all of these information-extraction approaches is the unavailability of adequate *context* (Pedersen and Bruce 1998). As shown previously in Fig.4, a computer-based *data-processing* environment does not allow for the representation of context. Therefore, in such an environment, 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 widespread 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 the ontology of a 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 relevant 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.


Computer: Some information representation.
COMPUTER AGENTS (TOOLS)
HUMAN AGENT
DATA
001001101
90% island
Tuesday
18:00
tropical
101110
INFORMATION
CONTEXT
"It often rains in the early evenings on tropical islands"
UNUSED ADDITIONAL COMPUTER HELD INFORMATION
CURRENTLY UNUSED COMPUTER HELD DATA
RELEVANT IMMEDIATE KNOWLEDGE
HUMAN/COMPUTER SHARED ADDITIONAL KNOWLEDGE
CURRENTLY UNUSED HUMAN/COMPUTER CAPABILITIES
UNUSED ADDITIONAL HUMAN HELD KNOWLEDGE
RELEVANT ADDITIONAL HUMAN HELD KNOWLEDGE

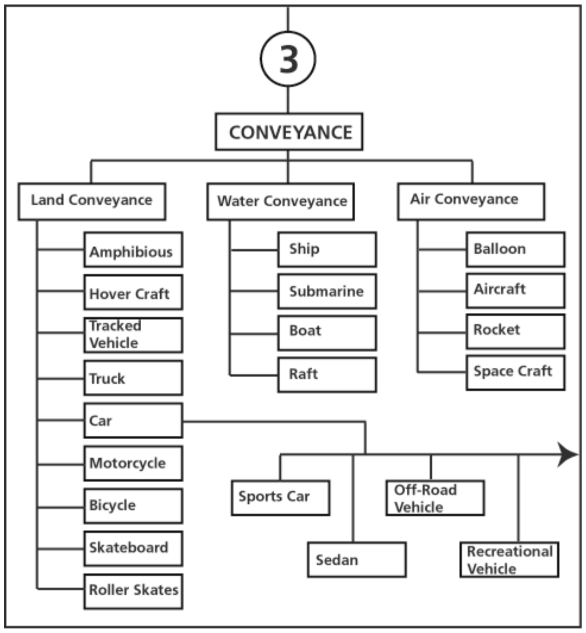
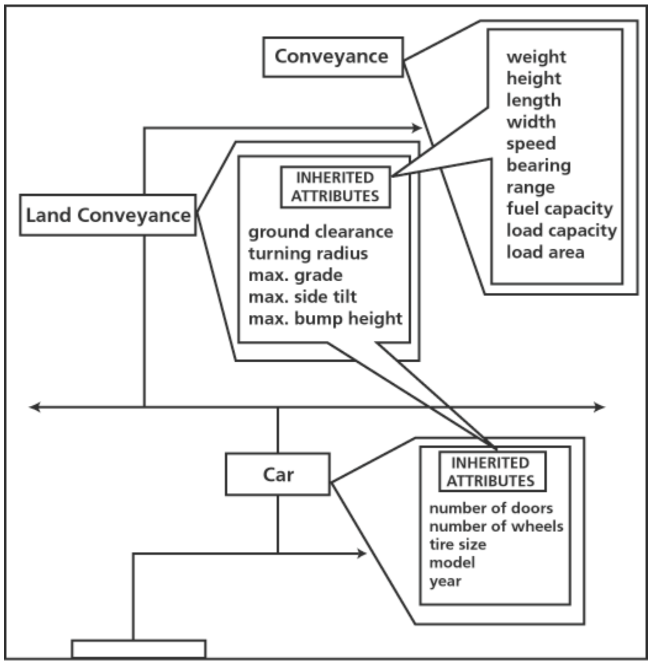


Fig.9: Early human-computer partnership

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.



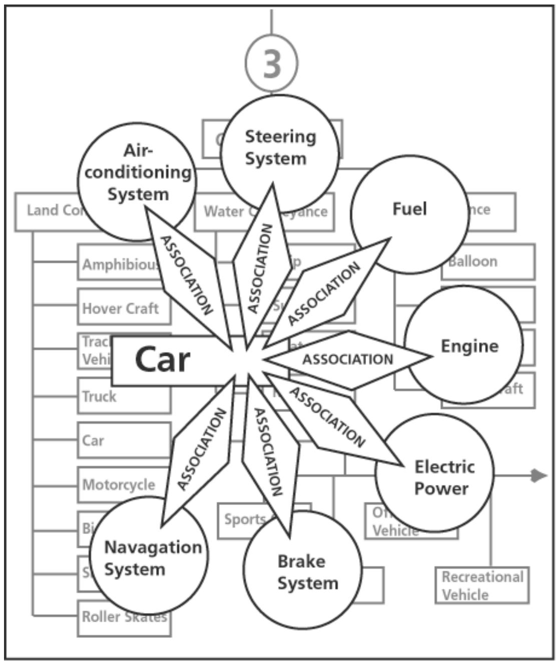


Fig.11: Object model - ***inheritance***

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.

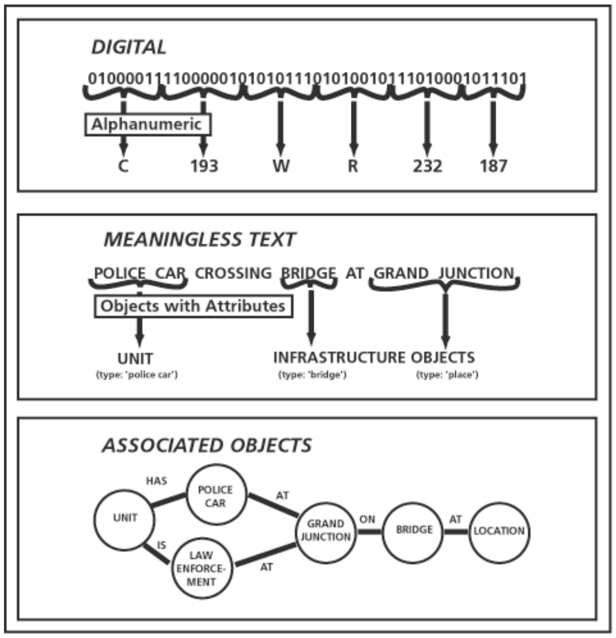


Fig.13: From ***digital*** to ***information***

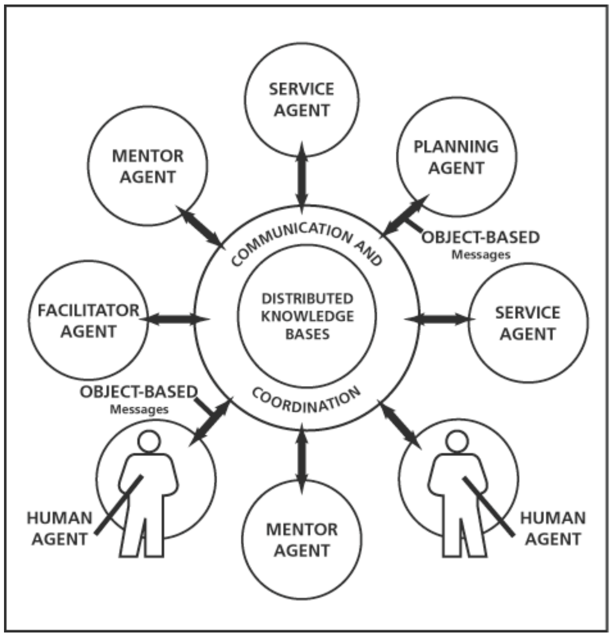


Fig.14: Types of agents

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.

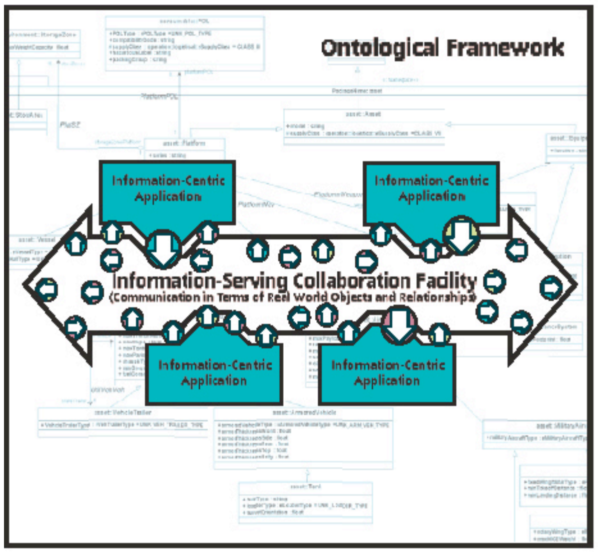
In this regard, the notion of *intelligent agents* refers to the existence of a common language (i.e., the ontological framework of information described earlier) and the ability to *reason* about the object characteristics and relationships embodied in the informational structure (i.e., rather than 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.


Ontological Framework
Information-Centric Application
Information-Centric Application
Information-Serving Collaboration Facility
(Communication in Terms of Real World Objects and Relationships)
Information-Centric Application
Information-Centric Application

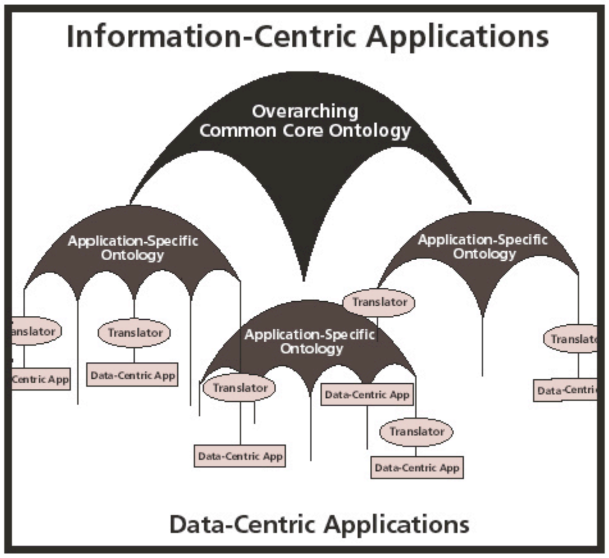

Information-Centric Applications
Overarching Common Core Ontology
Application-Specific Ontology
Application-Specific Ontology
Application-Specific Ontology
Translator
Translator
Translator
Translator
Translator
Translator
Data-Centric App
Data-Centric App
Data-Centric App
Data-Centric App
Data-Centric App
Data-Centric App
Data-Centric Applications

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 police patrol unit may request information relating to an unfolding emergency situation that may not be directly related to the current mission of that police unit, 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 computation 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 police patrol unit may have more than 100 objects included in its subscription profile. One set of these objects represents a *911* emergency call involving a stolen vehicle that is involved in a police chase. As the stolen car changes its position 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 goods movement tracking and security system such as SecureOrigins (Holguin 2003) and a ship load-planning system such as ICODES (Goodman and Pohl 2003)? With a focus on homeland security, SecureOrigins builds a profile of each goods shipment as it moves from origin to destination across international borders. Typically a container is fitted with an electronic security shield that may include active or passive GPS-based tracking devices. Taking into consideration all available data sources such as bank statements of the initial purchase, in-transit routes and events (e.g., delays, inspections and transshipments), shipper and receiver information, and national intelligence data, software agents determine the security risk associated with each container. Well before the shipment reaches the border SecureOrigins has already translated this risk assessment into a recommendation for the appropriate action by Customs authorities. ICODES, on the other hand, is concerned with the stowage of cargo on ships. Its agents assist the user in achieving the highest possible space utilization within the constraints imposed by hazardous material segregation rules, dimensional limitations (e.g., deck heights, openings, ramps, and elevator doors), crane capacities, and the trim and stability requirements of the ship. The objective of ICODES is to assist the user in the development of a near optimum load-plan and maintain in-transit visibility of cargo while it is being transported on ocean-going ships.

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 a 'truck' 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 domain of goods movement across international borders this general description (i.e., representation) of a 'truck' is further refined and biased toward a security role. In other words, the SecureOrigins application sees a truck as a mobile warehouse that needs to be protected from thieves and terrorists.

ICODES, on the other hand, sees a truck as a cargo item that takes up a certain amount of space and adds a given load to the trim and stability conditions of a ship.

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 Security Agent in the SecureOrigins application is able to advise appropriate agents in the ICODES application that a particular truck should be inspected before it is stowed on-board the ship because of a sequence of incidents that this truck was involved in prior to reaching the port. While not one of these events by itself might have been sufficiently serious to be noted as more than a minor anomaly, the combination of several incidents elevated these anomalies to a security risk. The agents are able to communicate across multiple applications at the information level through the common language of the ontological framework. Similarly, the SecureOrigins application is able to rely on the ICODES application to maintain in-transit cargo visibility, down to the location of a specific goods shipment in a particular truck on-board a ship en-route to a foreign country.

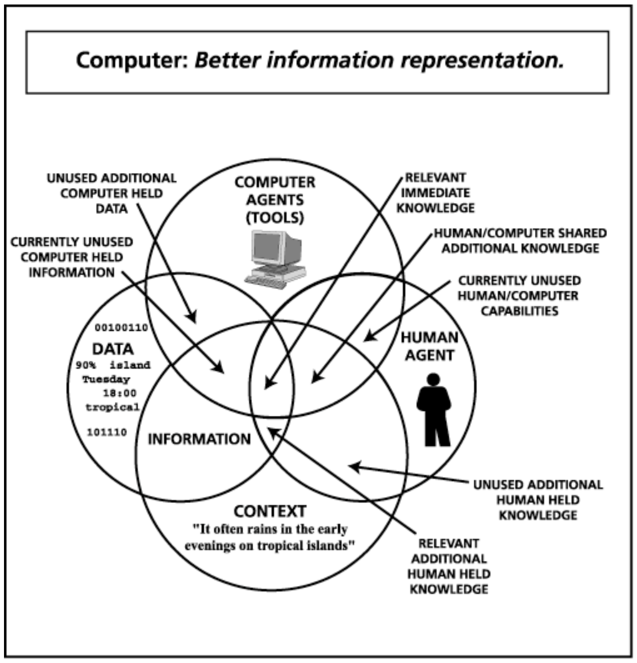
One might argue that this is all very well for newly developed applications that are by design implemented in an *information-centric* architecture, but what about the many existing *data-centric* applications that all perform strategic and indispensable functions? These existing legacy applications 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. Such a two-way mapping capability can be implemented in at least two ways, namely: as a universal translator that can be customized to a particular external application; or, as a branch of the internal information model of each application that models a bridging *view* of the other application. This type of *view* modeled inside an ontology is often referred to as a *façade* (see Section 6.2). In the case of a universal translator, 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.


Computer: Better information representation.
UNUSED ADDITIONAL COMPUTER HELD DATA
COMPUTER AGENTS (TOOLS)
RELEVANT IMMEDIATE KNOWLEDGE
HUMAN/COMPUTER SHARED ADDITIONAL KNOWLEDGE
CURRENTLY UNUSED COMPUTER HELD INFORMATION
CURRENTLY UNUSED HUMAN/COMPUTER CAPABILITIES
00100110
DATA
90% island
Tuesday
18:00
tropical
101110
HUMAN AGENT
INFORMATION
UNUSED ADDITIONAL HUMAN HELD KNOWLEDGE
CONTEXT
"It often rains in the early evenings on tropical islands"
RELEVANT ADDITIONAL HUMAN HELD KNOWLEDGE

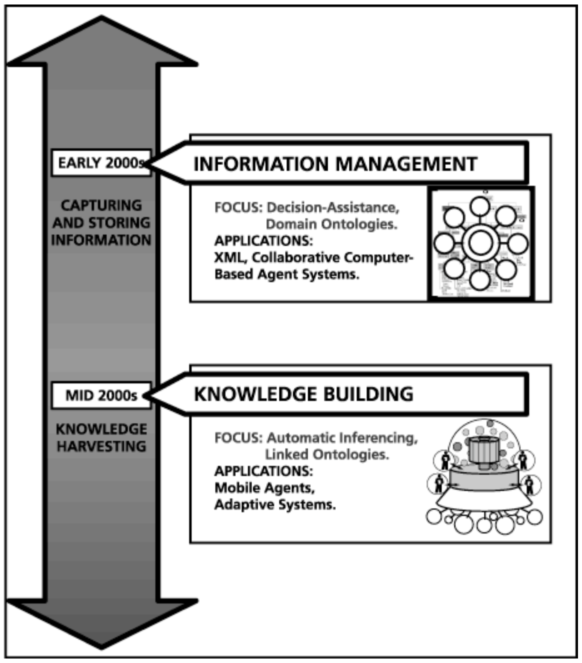

EARLY 2000s
INFORMATION MANAGEMENT
CAPTURING AND STORING INFORMATION
FOCUS: Decision-Assistance, Domain Ontologies.
APPLICATIONS:
XML, Collaborative Computer-Based Agent Systems.
MID 2000s
KNOWLEDGE BUILDING
KNOWLEDGE HARVESTING
FOCUS: Automatic Inferencing, Linked Ontologies.
APPLICATIONS:
Mobile Agents,
Adaptive Systems.

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 TIRAC™ Framework and Toolkit

The Toolkit for Information Representation and Agent Collaboration (TIRAC™) framework 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 TIRAC™ framework has been used with great success by CDM Technologies, Inc. and others for the design and development of commercial applications. Examples include SecureOrigins for international goods shipment tracking and security, PLANMAX for ship load planning and port management, and EMMERS for crisis coordination.

The core component of a TIRAC-based application is a virtual (i.e., ontology-based) 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 TIRAC-based applications the agents *collaborate* with each other and the human users in their monitoring, planning, and evaluation activities.

2.1 Architectural Features of TIRAC™

The TIRAC software development framework and architecture is based on the belief that representation is a fundamental key to the successful application of software-based reasoning. Specifically, the representation of context as opposed to fragmented data. Over the past decade there have been significant advances in the methods of representing semantics or meaning in domain-based context models (i.e., ontologies). Technologies such as the Resource Definition Framework (RDF: Brickley and Guha 2002), Ontology Inference Layer (OIL: Horrocks 2002, Gil and Ratnakar 2002), DARPA Markup Language (DAML: Cohen et al. 1998), Web Ontology Language (OWL: Dean et al. 2002), and the Unified Modeling Language (UML: Warmer and Kleppe 1999) offer many opportunities to represent the relationship-rich concepts, notions, and entities upon which meaningful software-based analysis can be based. While UML offers an extensive methodology for describing explicit relationship-rich context in an object-based manner, RDF, OIL, DAML, and OWL offer opportunities to represent more subtle relationships between concepts, notions, and entities that are based more on similarity than exact match. Regardless of representational approach, each of these technologies has the ability to represent relationship-rich context vital to the ability of software-based decision-support to reach the levels needed for multi-variable dynamic problem domains.

2.1.1 Technological Underpinnings

The successful and rapid processing of volumes of potentially relevant data (i.e., subject profiles, transactional events, etc.) into exploitable information-rich context requires technologies capable of mediating fragments of low-level data into contextually rich information. Technologies such as Extensible Stylesheet Language Transformations (XSLT) based transformation coupled with Extensible Markup Language (XML) descriptions are well suited to perform clearly definable translations. The more complex and potentially less obvious mappings, however, require a sophistication of the caliber inherent in expert system technology. In such environments, transforming or fitting collections of data into information-centric context may depend largely on the particular circumstances (i.e., variables) in which the transformation is to occur. In such cases, not only may extrapolation of data into meaningful information require intermediary knowledge to be developed but, in addition, the application of agent-based analysis may be needed to infer the final result based on domain (i.e., contextual) expertise. In other words, for the more complex transformations of data into useable information an entire agent-based decision-support environment may be appropriate. Architecturally, such a bridge may include an ontology representing intermediary representational targets along with communities of specialized collaborating agents applying their expertise to determine the context into which particular data fragments should be placed.

Once described within a semantically rich representation numerous opportunities exist to apply a variety of context-based analysis technologies to infer often subtle connections between seemingly otherwise trivial information and events. In addition, having access to such constantly evolving context allows the incremental identification of emerging patterns that are more significant than the sum of their parts.

In information-centric decision-support environments that are designed to support collaboration, presentation becomes an important issue. Users require the ability to not only survey a broad range of topics but also drill-down into relevant areas in an efficient and effective manner while still retaining a perspective of the overall operational picture. Further, access needs to be available on a wide range of platforms and at numerous locations. An emerging approach to this type of interaction model is the employment of Web portal technology (Daconta et al. 2003, 50). Being Web-based such portals can be configured for accessibility at any Internet-enabled location with appropriate security access. The main interface level is typically designed and configured to offer launching points into a variety of topical areas of interest. Once a topic is selected users are transported into an exploded view of that area of operations equipped with customizable tools to carry out focused activities. To prevent isolation from events outside the immediate focus Web portals can be configured to provide users with abbreviated views of highlights occurring in other areas of interest. In agent-enabled systems, such *mentorship* of user interests can be performed by a type of expert agent known as a *mentor agent*. The latter should be able to represent user interests by not only reacting to, but also initiating interaction with various resources (i.e., information sources, services, other agents, and even other users) on behalf of its owner.

Useful in the extensive and highly interconnected setting provided in Internet-based operating environments is the need to not only structure certain functionality as services but to also be able to discover such capabilities dynamically. Web service architectures provide standardized methods of both locating and engaging on-line services on an as-needed basis. Available services can be located in standard registries that provide descriptions of their capabilities along with their protocol for interaction. To exploit these facilities standards including the Universal Description, Discovery, and Integration (UDDI) protocol for registry-based lookup, the Simple Object Access Protocol (SOAP) for services interaction, and others have been developed allowing software processes (e.g., software agents) to dynamically discover needed services on-line and to pass relevant objects back and forth as needed. In this sense, the operating scope of what in classical software engineering has essentially been a predefined boundary can now be redefined in terms of the services that are being exploited at any given point in time. This type of architecture promotes a much more dynamic environment where the realm of available capabilities can evolve (i.e., enhanced and/or expanded) to meet current demands. Determination of such needs may even involve expert agents to appraise operational effectiveness as a measure of the availability of suitable services. Such operating environments become much more aligned with the notion of an evolving organism than a statically defined set of resources.

2.1.2 The TIRAC™ Framework

The TIRAC framework exists both as an architecture and as a set of development and execution tools that can be used to design, implement, and execute context-centric agent-based, decision-support solutions. The TIRAC model is based on a multi-tier architecture making clear and distinct separations between context, logic, and presentation (Gray and Lievano 1997). These tiers also represent the three major areas of support comprising the TIRAC model. The first of these areas, *context model management*, addresses access and lifecycle management of the data, information, and knowledge that is blueprinted by various context models. The second area, *logic*, addresses the housing and management of the business logic that operates over the evolving context. The third area, *presentation*, addresses the manner in which the context and logic is presented to both human and/or software-based users. Each of these tiers functions in an integrated fashion to form a comprehensive decision-support execution framework supporting agent-based reasoning over semantically rich context, as follows:

Context Tier: The *Information Server* consisting of collections of Object Servers constitutes the core of the Context Tier. These servers coordinate access and are responsible for the lifecycle management of collections of data, information, and knowledge that together form the context over which the other tiers operate. In addition, Object Servers promote location-transparency and distributed access and are compatible with supportive remote access technologies such as Remote Method Invocation (RMI), Common Object Request Broker Architecture (CORBA), and Java Bean technology.

To capitalize on the availability of the large number of information and service-related objects managed within the Context Tier, a powerful set of collaborative services is available to perform focused searches. In addition, these services keep interested parties current in terms of relevant object-level changes. Within a TIRAC operating environment the former of these two services is provided by a context-sensitive query service. The context-sensitive aspect of this service is exposed as a query language that supports the formulation of expressions directly in terms of relevant context models (i.e., ontologies) on which potential results are based. This type of client-initiated content retrieval is effective when relevant changes have occurred and actual context values are desired.

In contrast to the query service, the second collaborative service supports currency among interested clients without requiring clients to initiate retrieval. Through a subscription service, clients can register various interest profiles that are subsequently monitored for satisfaction. Like the expressions supplied to the query service, client interests are described in the direct terms of the relevant context model properties. In this manner, clients are able to have a subscription service monitor potentially complex events that directly represent operational conditions. Upon satisfaction of a pre-registered interest the subscription service notifies interested clients. Herein rests the second optimization available to subscription service clients. In many cases the relevant aspect of an interest is not so much the details (i.e., the values) of the event but the fact that the event has occurred and that the interest has been satisfied. Clients wishing to obtain particular context values may specify the desire to have such details accompany the initial notification on an interest-by-interest basis. In cases where the need for details cannot be pre-determined, clients may employ the query service as a follow-on activity. For extensive conditions, the choice of not paying the processing and communication overhead associated with sending event values can translate into notable performance increases. The opportunity for this type of optimization is directly enabled by the ability of clients to be more precise in describing their particular interests. In a less descriptive environment, clients would be typically forced to interrogate events in an effort to determine the type of subsequent action required.

Logic Tier: Expert agents within a TIRAC-based operating environment can take a number of forms including procedural, rule-based, and algorithmic. Regardless of paradigm and applicable technology, engines housing and managing communities of agents integrate with the TIRAC Context Tier following the standard TIRAC connection pattern (i.e., adherence to the standard access interface and able to *speak*, or in some cases *discover*, the particular context language being offered). While this open architecture approach supports the creation of any number of supportive logic engines the TIRAC suite offers a fully functional implementation of a reasoning-enabled inference engine, referred to as the *Agent Engine* within the TIRAC framework.

Adhering to the standard TIRAC connection pattern, instances of this type of logic engine exist as clients to the Context Tier and as such have available to them the standard Context Tier services (i.e., access, query, subscription, and persistence). As a means of connecting rule-based agents housed within the Agent Engine with the rest of the decision-support system, the Agent Engine automatically registers a personal interest profile for each agent with the Context Tier's subscription service. These profiles are based on the specific changes in context that trigger the activation of the particular agent's rule sets. In this manner, Jess-based agents are seamlessly integrated with the relevant changes in context occurring as overall system execution progresses (Freidman-Hill 2003). Like any Context Tier client, once an agent's rule has been executed, further context can be obtained through either engaging the query service or directly accessing known context objects. As was stressed in the discussion of the Context Tier interactions, whether describing particular interests or drilling down into a specific section of context, takes place in terms of context objects. This again illustrates the degree to which a context-centric representation is preserved as information is processed throughout the decision-support environment.

2.1.3 A View-Based Approach to Context Segregation

While TIRAC's explicit (i.e., the TIRAC Agent Engine) and implicit (i.e., a standard tier interaction interface upon which various logic engines can be built) facilitates a variety of approaches to the manner in which differing versions of operational reality are separated from one another, significant success has been achieved by the TIRAC user-base involving the notion of a *view*.

A *view* in this sense can be thought of as a particular perspective, perhaps among many, of some meaningful scenario. 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. For example, in the application domain of disaster coordination and management a single *view* may be used to represent the information and events occurring in the disaster area in support of first responder operations. In a similar manner, any number of additional *views* may be usefully employed 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 *view* and an agent community (Fig.19). This means that independent of exactly which version of events a *view* represents, there exists a dedicated community of agents whose sole perceptual scope is bound to that *view*. Organizing alternative scenario analyses in this manner allows for an efficient and effective means of distinguishing all context and collaboration 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 context values and resulting collaboration is completely disjoint from the other.

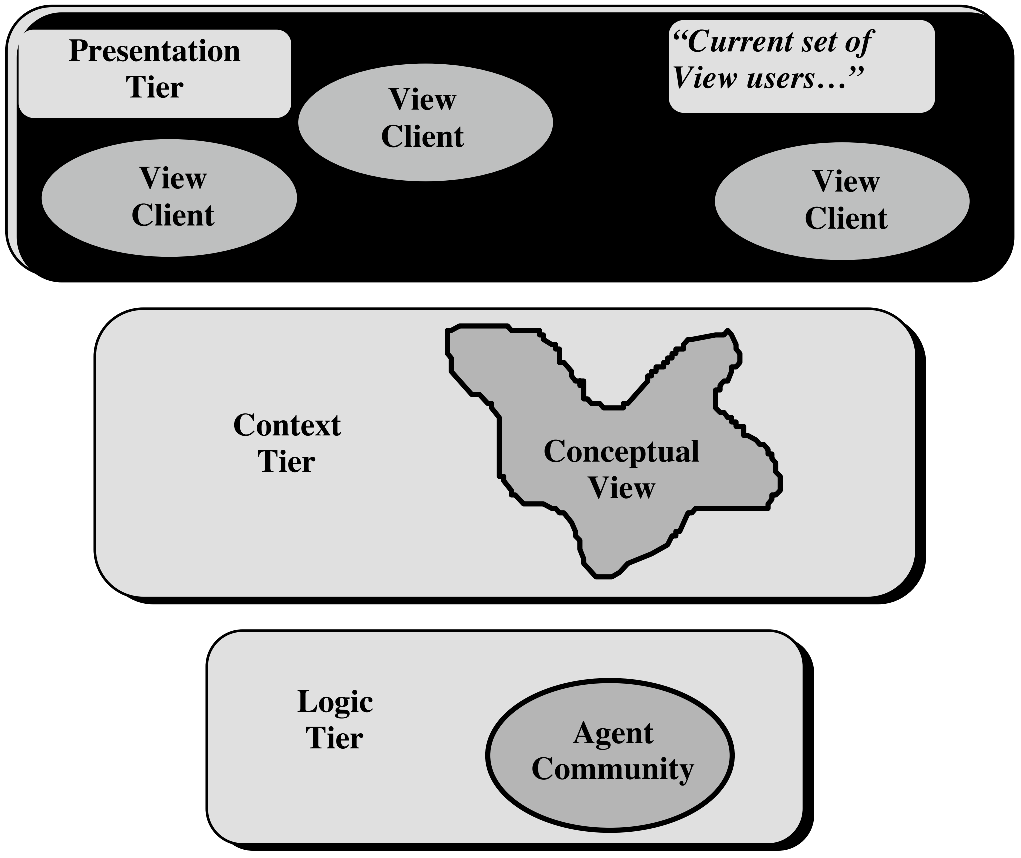


Fig.19: **Segregation of alternative scenarios as *Views***

Presentation Tier: The TIRAC toolset supports presentation in two distinct forms. The first of these targets graphical presentation to human users. In this case, support consists of several useful self-managing *controls* offering reusable utility for common GUI functionality in agent-based decision-support systems (e.g., agent status and reporting, calendar management, report management, scenario recording and playback, context inspection, map display, etc.). Developers can exploit these tools to rapidly produce useful presentation clients. A key enabler for this integration is the Object Manager, or OM. A client-side layer, the OM both abstracts out complexities associated with distributed object access as well as offering a means for discovering various properties of the context tier. The former of these features is especially useful for clients that wish to *see* context as string-based symbols. GUI clients typically fall into this category as well as some symbolic inference engines. The latter of these features is the beginning of TIRAC's intent to support complete context discovery at runtime. Using discoverable context element templates, this facility supports the structural discovery of context in terms of element attributes and other structural properties. However, understanding that structure is only one part of true context discovery. OM to date does not attempt to support any type of semantic discovery. Standardized exchange languages such as XML are also limited to conveying structure and not semantics. While structure can convey the organization of a chapter in a book it does not convey what a chapter *is* or *implies*. It is well understood in industry and academia that semantic discovery offers a more significant challenge and is currently the subject of considerable research.

The second area of TIRAC's support for the presentation tier addresses presentation to software-based users, or external systems. The key issue in such support is the potential differences in representation between systems. In other words, similar to a GUI offering a graphical representation of the context to human users, presenting to a software-based user may also require some level of translation to a representation more native to the receiver. TIRAC addresses this need by offering a type of transformation engine. This engine can be used to effectively construct architectural bridges between systems operating over differing representations. Bridging, in this sense, not only supports functional connections but also connections at the representational level, a level critical in context-centric systems where much of the ability to interoperate with external systems depends on the richness and clarity of informational/data feeds.

In its isolated form the TIRAC Translator supports transformation of instances based on one XML schema to instances based on another XML schema. For the more straightforward mappings between schemas the Translator offers an XSLT-based mapping engine capable of applying XSL transforms to XML objects. XSLT rules can be defined at development time or, for more dynamic transformation, constructed on-the-fly during execution. However, even with the ability to define transformation rules at run-time, XSLT rules operate in a fairly isolated manner. For more complex transformations requiring a deeper degree of analysis to determine the appropriate end-result of a transformation, a more powerful technology is required. For such cases, an environment with inference-based analysis can be applied to determine the appropriate context into which the input is to be transformed. Such analysis could take the form of a micro decision-support environment involving collaboration among communities of rule-based agents perhaps progressively building the *transformation solution* through employing several intermediary context models. While transformations of this magnitude would typically be performed as a follow-on activity to importing information from another system, the inference engine based facility of the TIRAC Translator offers opportunities for architecturally encapsulating potentially complex transformation activities into a well-discernable *bridge* thus resulting in a simplified and more manageable translation architecture.

Integration of the Translator with its clients (i.e., producers and consumers) adheres to the *adapter* architectural design pattern. Referred to as *connectors*, it is these components that essentially *adapt* the XML schema-based interface supported by the Translator to the specific interface native to the particular client. Such connection typically involves interface-specific calls to the native interface for the purpose of both retrieving and contributing transformation content. For example, in the case where the interface of a Translator client is Structured Query

Language (SQL) based, the associated connector is responsible for formulating and issuing the appropriate SQL statements to both select *(obtain)* and insert native content depending on the direction of transformation. If not already in XML form, both of these activities involve repackaging content into and out of the client's native format. Note, this repackaging should not be confused with the much more complex activity of content transformation executed by the Translator's mapping engine(s), potentially involving significant changes in content representation in support of an alternative system perspective. Rather, this activity simply reformats values in terms of an XML schema native to the particular system's representation. Again, this is a repackaging of content as opposed to the transformation of content.

Capitalizing on the code generation capabilities supported by the TIRAC toolset, the connector responsible for adapting the Translator to a TIRAC-based client system is automatically provided. The *front-end* XML schema for the TIRAC-based Translator client is automatically built based on the structural properties of the Context Tier. The connector functionality is then configured at run-time based on the structure of the content housed in the TIRAC-based Context Tier. In this way, only connectors adapting to non-TIRAC systems need to be manually developed. As mentioned earlier, support for automatic model-driven implementation is a core theme in the TIRAC development toolset.

2.2 The TIRAC™ Software Development Process

The TIRAC™ 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 TIRAC™ 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-based 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 TIRAC tool-set.

2.3 Origin, Proprietorship, and Licensing

The initial formulation and development of TIRAC™ 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 TIRAC™ 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 TIRAC-based projects for commercial and non-military[1] government customers. 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 TIRAC 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 TIRAC. Accordingly, in the event that at some future date CDM should cease to exist, licensees will continue to have assured access to TIRAC expertise, development capabilities, and support through the CADRC as an official Cal Poly University Research Center.

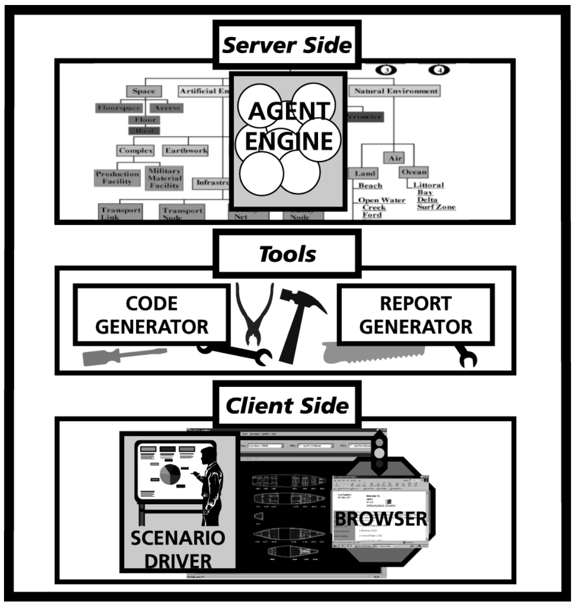
The TIRAC toolkit can be licensed directly from CDM Technologies, Inc. together with variable software maintenance contract terms including technical support and consultation services. The TIRAC toolkit consists of the following principal components and a larger number of secondary accessory tools and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

- TIRAC™ object-based schema implementation.
- Library of TIRAC™ code and report generating tools.
- TIRAC™ information server.
- TIRAC™ persistence service.
- TIRAC™ agent engine.
- TIRAC™ client user-interface components.

[1] The companion Integrated Cooperative Decision Making (ICDM) toolkit is licensed by CDM Technologies for military software systems (Pohl et al. 2004).

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 its commercial arm CDM Technologies, Inc. (CDM), both in San Luis Obispo, California, 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 suit of tools is known as the Toolkit for Information Representation and Agent Collaboration (TIRAC™) software development framework (Fig.20).



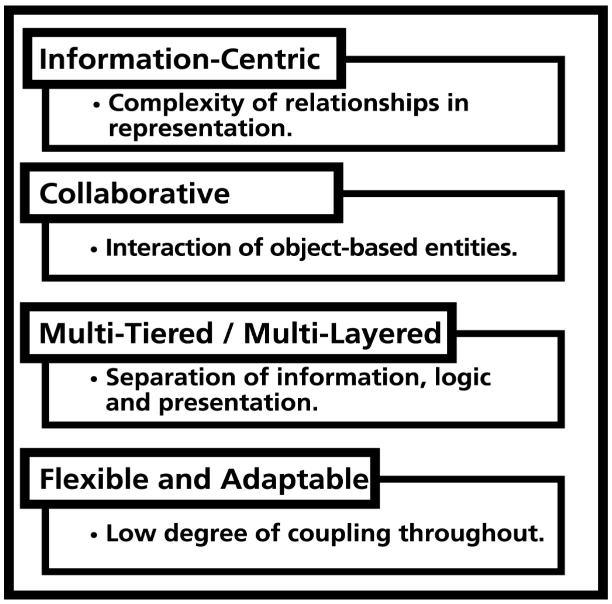

Information-Centric
• Complexity of relationships in representation.
Collaborative
• Interaction of object-based entities.
Multi-Tiered / Multi-Layered
• Separation of information, logic and presentation.
Flexible and Adaptable
• Low degree of coupling throughout.

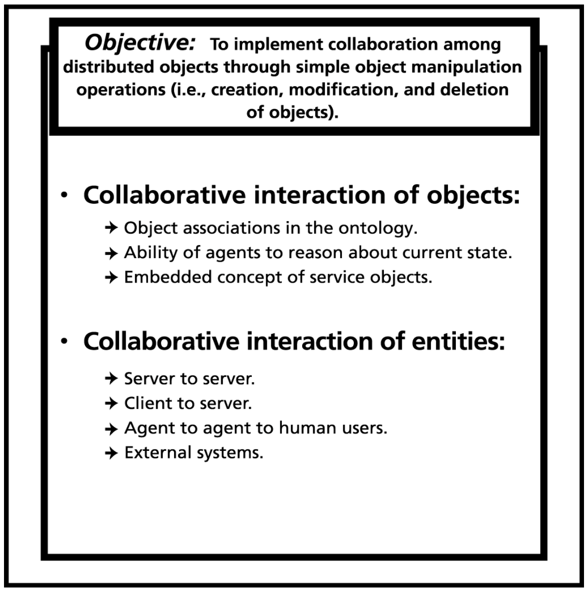
Fig.20: TIRAC Development Toolkit

Fig.21: TIRAC Design Principles

Not only does TIRAC 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 TIRAC is founded; namely, collaboration-intensive, context-based representation, flexibility and adaptability, and multi-tiered, multi-layered architecture (Fig.21).

3.1 Collaboration-Intensive

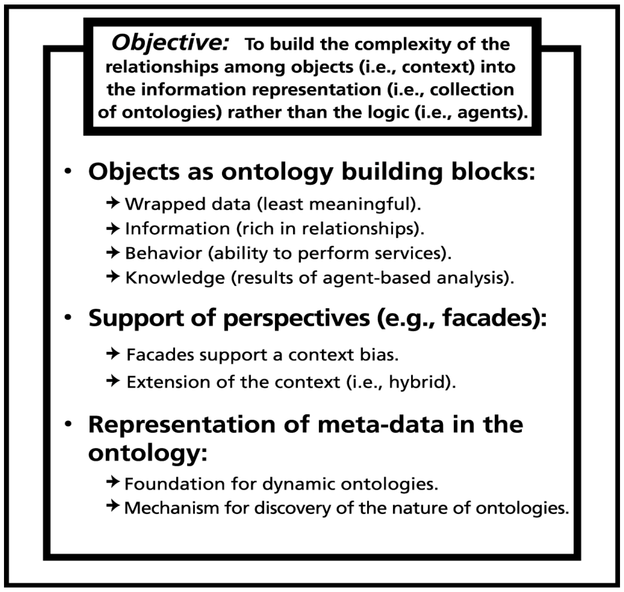
Certainly in the real world, collaboration among decision-makers and experts is a critical ingredient in making educated and effective decisions. This is especially true when operating across an extensive and varied set of domains. Through years of research in collaborative design the CADRC has found 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.



Objective: **To implement collaboration among distributed objects through simple object manipulation operations (i.e., creation, modification, and deletion of objects).**

- **Collaborative interaction of objects:**
  - → Object associations in the ontology.
  - → Ability of agents to reason about current state.
  - → Embedded concept of service objects.
- **Collaborative interaction of entities:**
  - → Server to server.
  - → Client to server.
  - → Agent to agent to human users.
  - → External systems.

Fig.22: Design Principle: *Collaboration*



Objective: **To build the complexity of the relationships among objects (i.e., context) into the information representation (i.e., collection of ontologies) rather than the logic (i.e., agents).**

- **Objects as ontology building blocks:**
  - → Wrapped data (least meaningful).
  - → Information (rich in relationships).
  - → Behavior (ability to perform services).
  - → Knowledge (results of agent-based analysis).
- **Support of perspectives (e.g., facades):**
  - → Facades support a context bias.
  - → Extension of the context (i.e., hybrid).
- **Representation of meta-data in the ontology:**
  - → Foundation for dynamic ontologies.
  - → Mechanism for discovery of the nature of ontologies.

Fig.23: Design Principle: *Information-Centric (Context)*

Whether human or software-based, collaboration within a TIRAC-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 represent 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 a TIRAC-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.22). 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.

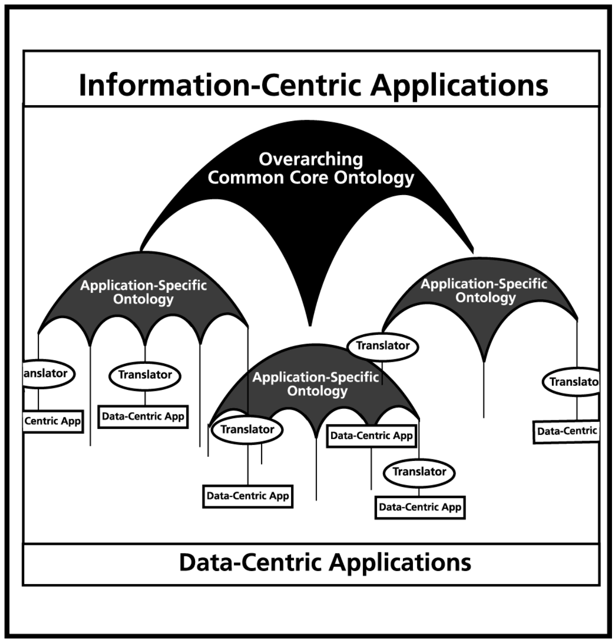
3.2 Context: Information-Centric

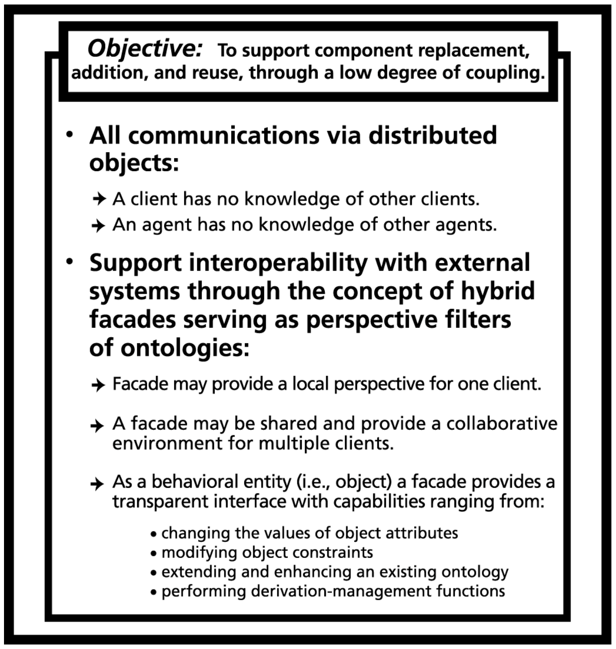
Representation can exist at varying levels of abstraction (Fig.23). The lowest level of representation considered in this paper is *wrapped* data. Wrapped data consists of low-level data, for example a textual e-mail message that is placed inside some sort of an e-mail message object. While it could 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 TIRAC 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 TIRAC 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, TIRAC-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.24). 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.


Information-Centric Applications
Overarching Common Core Ontology
Application-Specific Ontology
Application-Specific Ontology
Application-Specific Ontology
Translator
Translator
Translator
Translator
Translator
Translator
Centric App
Data-Centric App
Data-Centric App
Data-Centric App
Data-Centric App
Data-Centric
Data-Centric Applications



Objective: **To support component replacement, addition, and reuse, through a low degree of coupling.**

- **All communications via distributed objects:**
  - → A client has no knowledge of other clients.
  - → An agent has no knowledge of other agents.
- **Support interoperability with external systems through the concept of hybrid facades serving as perspective filters of ontologies:**
  - → Facade may provide a local perspective for one client.
  - → A facade may be shared and provide a collaborative environment for multiple clients.
  - → As a behavioral entity (i.e., object) a facade provides a transparent interface with capabilities ranging from:
    - changing the values of object attributes
    - modifying object constraints
    - extending and enhancing an existing ontology
    - performing derivation-management functions

Fig.24: Ontology Cluster Supporting Multiple Levels of Abstraction

Fig.25: Design Principle: *Extensibility and Adaptability*

3.3 Extensibility and Adaptability

One of TIRAC'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. TIRAC 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.25). There are two key TIRAC 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. TIRAC 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, TIRAC promotes the incorporation of such translation into the ontology itself. In other words, using TIRAC'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 TIRAC adheres addresses the architectural organization of TIRAC-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 a TIRAC-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.26).

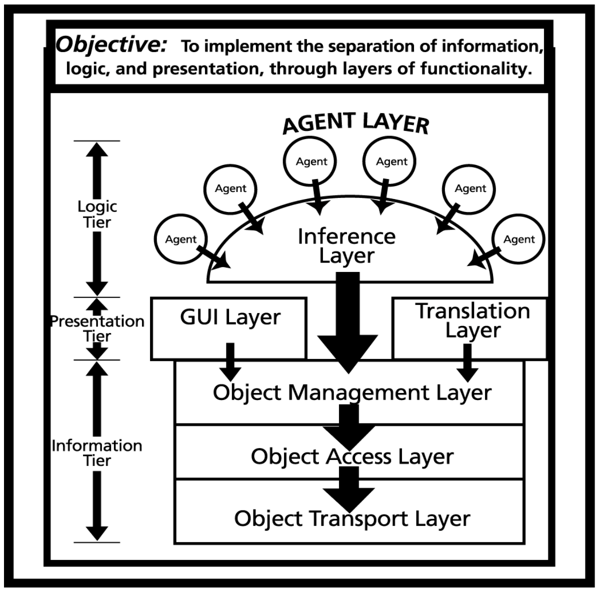


Fig.26: Design Principle:
Multi-Tiered, Multi-Layered Architecture

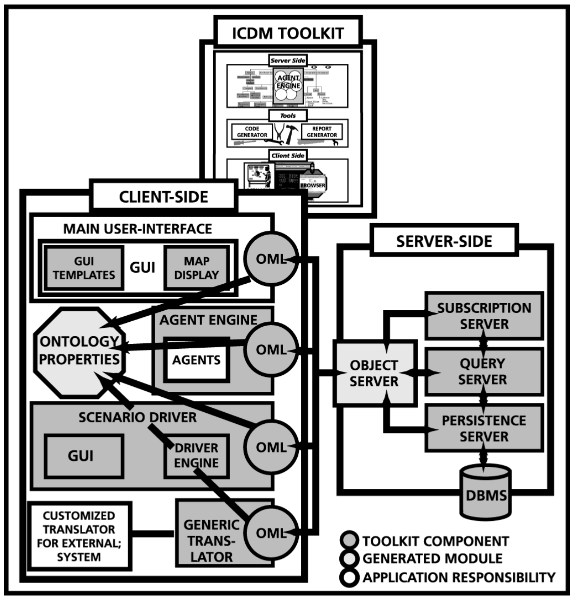


Fig.27: TIRAC-Based System Component Types

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 TIRAC 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 TIRAC-Based Application Environment

As a toolkit for the development of agent-based, decision-support systems TIRAC 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.27). 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, TIRAC can be effectively utilized in the rapid development of agent-based decision-support systems spanning a variety of complex domains. TIRAC 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. 1989 and 1991; Pohl K. 1996)) to the security of goods shipment across international borders (e.g., SecureOrigins (Holguin 2003)). 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. An Ontology-Based Subscription Service

As mentioned previously in Section 2.1.2, the TIRAC model is based on a three-tier architecture that draws a distinction between information, logic, and presentation (Gray and Lievano 1997). These tiers are represented by the three major components comprising the TIRAC model: the *Information Tier* consisting of a collection of information management servers; the *Logic Tier* represented currently by an Agent Engine; and, the *Presentation Tier*.

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.

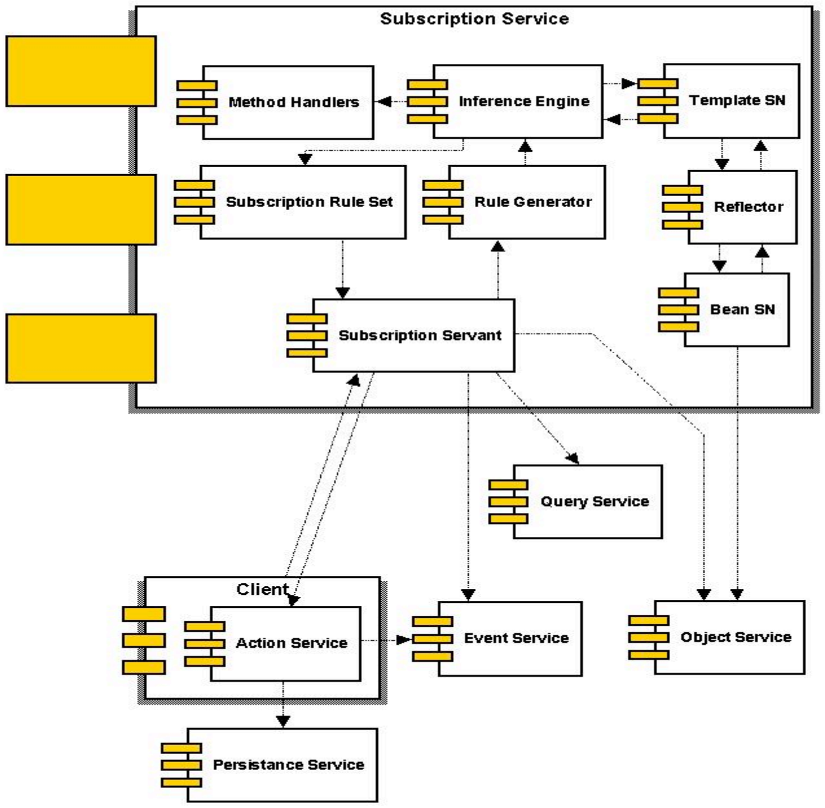


Fig.28: Inference engine-based subscription service architecture

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 choose 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.

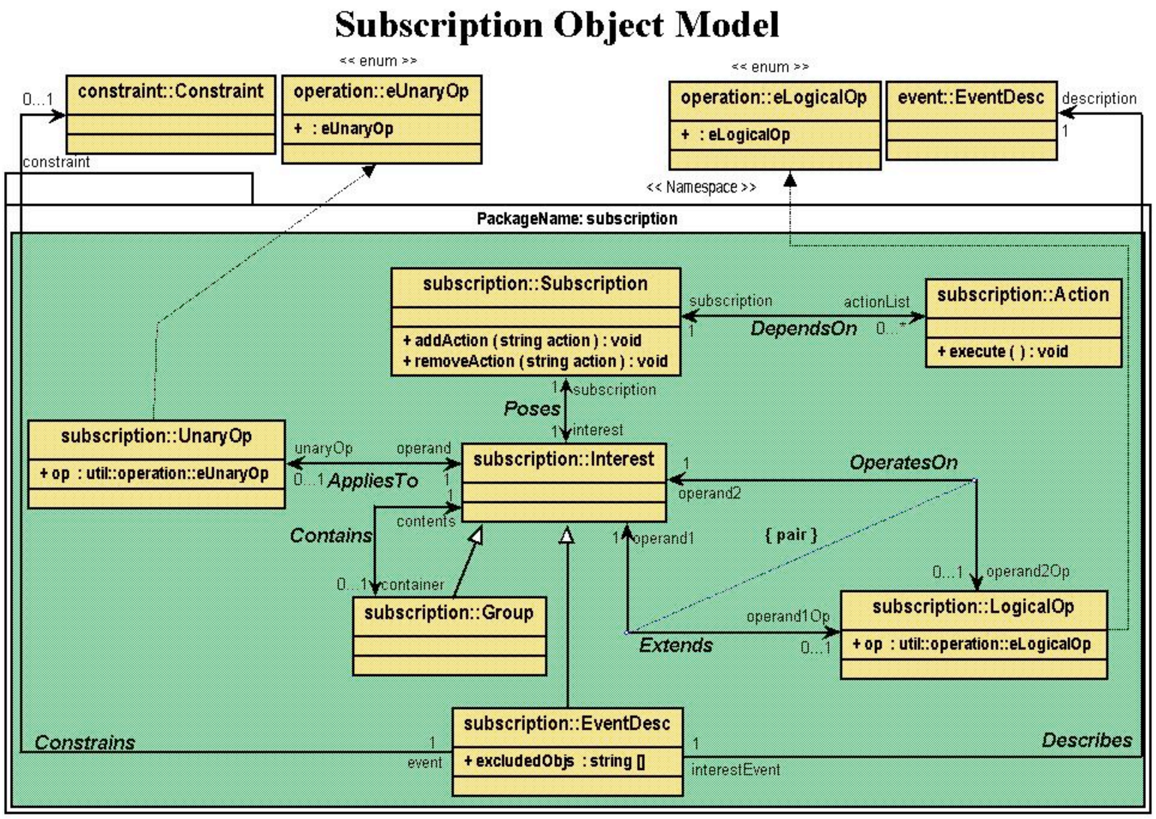


Fig.29: Subscription service domain ontology described

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.

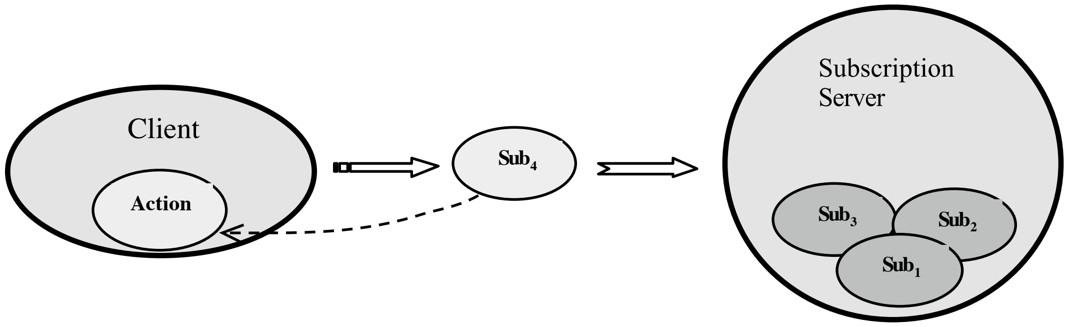


Fig.30(a): Subscription registration

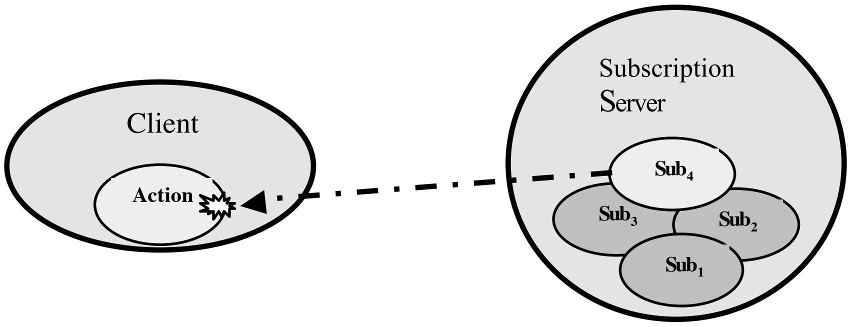


Fig.30(b): Client notification regarding interest satisfaction

5. TIRAC™ in Practice: *SecureOrigins* as an Example

This Chapter describes the principal components of the TIRAC software development framework using a particular ontology-based, multi-agent application as an example. The application chosen for this purpose is SecureOrigins, which is focused on the security of goods shipments that cross international borders.

5.1 The SecureOrigins Problem and Solution Domain

In an effort to address trafficking and security vulnerabilities at international border crossings in a manner that capitalizes on current and emerging technologies, CDM Technologies in conjunction with Holguin Group, EVP Group and others embarked upon the development of the TIRAC-based SecureOrigins software system. An ontology-centered multi-agent system architecture was considered to be a mandatory choice because of the complexity of the multi-variable homeland security problem environment and the need for intelligent tools that could dynamically adapt to changes in smuggler and terrorist strategies.

5.1.1 The Goods Shipment Security Problem

Since September 11, 2001 the United States (US) has been faced with a serious goods movement security dilemma: how to prevent the smuggling of nuclear, biological and chemical weapons into the US without seriously impacting the flow of shipments across its borders and through its ports? Realistically, every container entering the US must be viewed as a potential weapon of mass destruction, every conveyance carrying the container as a delivery device, and every port of entry and inland destination as a potential target. With individual ships capable of carrying up to 6,000 containers each, more than six million container shipments entered the US in 2001. At an approximate value of $60,000 per container, this constitutes an annual trade volume of around $360 billion.

Physical intrusive inspection of more than a very small percentage (i.e., less than 3%) of container shipments at the port of entry would have serious national economic implications and is therefore not practical. From a business perspective, speed and cost are the critical criteria for the selection of transportation routes and shippers. While logistic costs have declined from 20% to about 15% of the Gross Domestic Product over the past two decades, the inventory (i.e., goods in storage) cost still amounts to some $400 billion. The potentially serious impact of increasing the rate of full physical inspection at the point of entry is easily demonstrated by the following example. Manual intrusive inspection of a 3,000 container ship (assuming an optimistic inspection period of only 60 minutes per container) would add at least two man-years of labor to the unloading operation. Yet, an additional 1% burden on logistic costs (e.g., increased inventory due to slower processing) would cost the US economy around $25 billion (or the equivalent of 15,000 lost jobs) annually.

Such a security problem exists at the US-Canada and US-Mexico borders. For example, El Paso (TX) and Juarez (Mexico) are twin border communities located in close proximity to each other on either side of the US-Mexico border. Over the years five major industrial parks have formed on the Mexican side of the border, driven to a large extent by US commercial interests. Many US companies have found it profitable to locate manufacturing and assembly plants in Mexico and take advantage of significantly lower labor costs. This cross-border commercial relationship generates a great deal of goods shipment traffic across the border.

On any given day approximately 3,000 trucks transport goods and parts to and from the Juarez industrial parks across the US-Mexico border. These trucks together with other vehicular traffic pass through three congested Border Check Points. Waiting times for trucks can exceed three hours, leading to loss of time, increased shipping costs, environmental pollution, and frustration. However, even more importantly the current shipment control processes do not assure a reasonable level of security. There is an urgent need for an information system solution to support a greatly improved border control process that will provide an adequate level of security with a minimum impact on the flow of cross-border vehicular traffic. Typically, the principal elements of such a solution should include the following:

1. Electronic capture of reliable and complete shipment documentation at the point of departure through: agreements with shippers, goods owners and local Mexican authorities; sealing of shipments with approved mechanisms and devices; wireless electronic data entry devices; barcodes and smart tags (i.e., RF-Tags); Web-cameras; and data integration within a shared database facility.
2. Tracking of shipments utilizing wireless GPS devices, automatic electronic sensing gates, and Web-camera surveillance facilities.
3. Near real-time processing of shipment information by collaborative, intelligent agent tools to identify anomalies and select higher risk shipments for full inspection at the point of entry (i.e., at the Border Check Point). In addition, selection of random shipments for full inspection at the point of entry.
4. Automatic identification of shipments and drivers at the point of entry through bar codes, smart tags, Web-cameras, and other non-invasive technology applications.
5. Thorough inspection of selected shipments, in designated areas, utilizing physical inspection technologies.
6. Continuous shipment information tracking to the completion of each shipment transaction.
7. Automatic maintenance of historical records of shipments to serve as a basis for analysis, trend identification, and 'what if' explorations.

Over the past several years goods movement security at US ports and international borders has increasingly focused on two strategies as a countermeasure. The US Customs Service has been exploring ways of extending its 'trusted shipper' (i.e., 'trusted source') program. This program is essentially based on the concept that systematic security measures executed by a compliant and reliable shipper at the point of departure will not require inspection at the point of entry. Under this first strategy the Customs Service defines a set of requirements that a 'trusted shipper' must adhere to, and then strives to the best of its ability to verify that the shipper is and continues to be compliant with these requirements. The second strategy relies on advances in non-intrusive inspection technologies (Mallon 2001). Such technologies include neutron and gamma ray scanning, motion detection, as well as radiological and chemical sniffing devices. Typically, this kind of inspection can be performed in seconds while the container is in motion between gates, or in one or two minutes in a special drive-through en route test station (Fig.31). While this second strategy is certainly attractive and promising it is as yet expensive, the necessary devices are not available in the required quantities, and the technology lacks proven reliability.

Clearly, any control action at the point of entry, such as inspection, has only a small probability of preventing a determined terrorist or contraband act. Accordingly, it is now well recognized that the interdiction of terrorist activities in the realm of goods movement into and out of the US cannot start at the border, but must commence with the shippers and their customers and the collection of the data representing the physical and transactional beginning of the goods acquisition chain. In this respect goods movement security is firstly an information technology problem and only secondly a physical border and port problem (Quartel 2002).

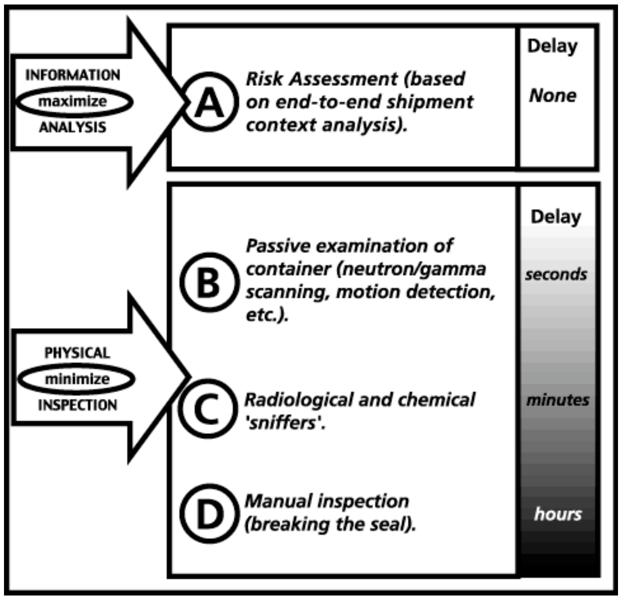

INFORMATION
maximize
ANALYSIS
A
Risk Assessment (based on end-to-end shipment context analysis).
Delay
None
PHYSICAL
minimize
INSPECTION
B
Passive examination of container (neutron/gamma scanning, motion detection, etc.).
C
Radiological and chemical 'sniffers'.
D
Manual inspection (breaking the seal).
Delay
seconds
minutes
hours

Fig.31: Inspection delay levels

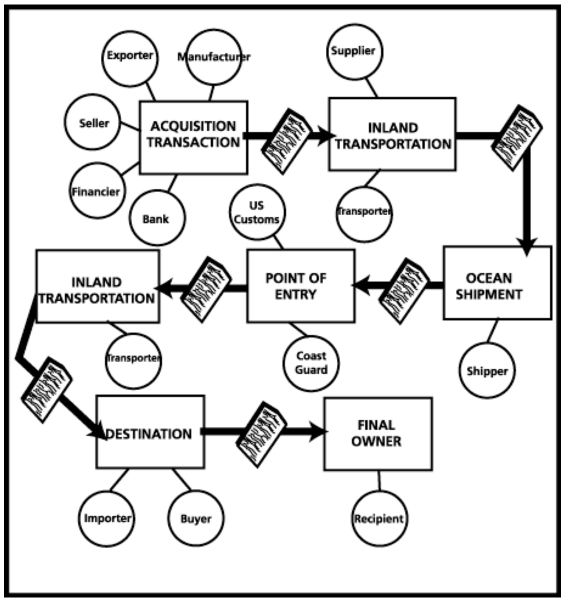


Fig.32: End-to-end shipment process

5.1.2 The *Profiling* Solution Approach

The most promising information technology solution approach to goods movement security relies on the concept of *profiling* shipments. In other words, it attempts to capture the end-to-end context of each shipment so that software agents operating in a collaborative mode can automatically perform at least the initial filtering, evaluation and profiling functions. In general terms the necessary context includes the data used by commerce to achieve efficiency, the data obtained through the in-transit monitoring of shipments and the detection of apparent anomalies, the relevant law enforcement data on both sides of the border, and at least some elements of national security intelligence.

The end-to-end supply chain for a complete shipment transaction commences with the initial acquisition of the product to be shipped to the US (Fig.32). For example, the acquisition transaction data that is relevant to the construction of the shipment profile includes not only the purchasing documents, but also the bank statements that describe the financial transactions and even the documents that define the financing agreement itself. Certainly the monitoring and tracking data related to the ocean voyage and the inland transportation on both sides of the border is of high value, particularly if the shipment process is based on 'trusted source' concepts.

Finally, data related to the destination and the final owner constitute important contributions to a complete shipment profile. Relevant data collection questions include the following:

- What kind of cargo does this shipment consist of?
- Where did the shipment originate?
- Who purchased these goods?
- Who sent the shipment?
- Where is the shipment going?
- Who are the shippers on both sides of the border?
- What was the planned and actual shipping route?
- How long has the shipment been in transit?
- Who will receive the shipment?
- What is the history of all parties who touched the shipment?

The focus must be on profiling the shipment first, and then the container. Many of the required data elements are already available in existing documents such as: the shipper's Letter of Instructions; the various commercial invoices; the Certificate of Origin; and, the carrier's Bill of Lading. Additional data elements that are needed for building a complete and reliable shipment profile include: financial data such as letters of credit and bank reports; inland transportation records from both sides of the border; and, in-transit monitoring data for identifying transshipment route changes, delays, and other events.

In summary, the following guiding principles constitute the overall solution framework within which the SecureOrigins design requirements were formulated:

1. Move the security process to the source of the shipment and provide incentives for complete, timely, and accurate data.
2. Provide physical protection of the in-transit shipment and transportation infrastructure (e.g., sealing standards, tracking, and in-transit visibility).
3. Streamline the point of entry processing, including the provision of express lanes, shipper incentives, and indemnification options.
4. Implement shipment and container profiling through data capture and the application of intelligent information management technology.
5. Automatically select higher risk shipments and containers for the appropriate level of inspection, by taking advantage of collaborative software agents operating in an information-centric decision-support system environment. At the same time select random inspection samples and dynamically modify the profile building rules used by the software agents based on inspection results.
6. Progressively improve the physical inspection hardware devices with the objective of increasing reliability, and decreasing inspection time and cost.

It must be recognized that ports and border checkpoints are part of the last line of defense. In other words, *the decisions to inspect or not to inspect a container and the level of inspection should be made well before a shipment reaches the port of entry*. This is possible only if the full context of the shipment is available for consideration.

5.2 The TIRAC-Based *Secure Origins* Architecture

Conceptually, the technical approach utilized by the SecureOrigins system solution applies intelligent agent technology to provide a shadow staff of digital assistants to responders and coordinators at all nodes within an extended, distributed, intelligent homeland security network. These assistants analyze and categorize incoming signals and data, and then issue warnings and alerts as appropriate. They manipulate the incoming data within an internal information-centric representation framework to publish statements of implication, and if so empowered, proceed to develop plans for appropriate action.

The need for SecureOrigins to support near real-time interactions with both internal and external components was recognized early during the design stage. The digital assistants or agents must be able to receive status reports, track shipments, incorporate suitable and available assets in plans, and provide appropriate updates on location and security risks. At the same time human users need to be able to interact with the agents and also initiate action sequences, request reports and enter data on their own accord. Finally, it is necessary for SecureOrigins to be able to interface with existing data-centric systems and place any data received from these external systems into the context provided by its own internal information model (i.e., ontology).

TIRAC supports these kinds of interactions through two distinct interface facilities. The first of these targets graphical presentation to human users. In this case, support consists of several useful self-managing *controls* offering reusable utility for common Graphical User Interface (GUI) functionality such as agent status and reporting, calendar management, report management, scenario recording and playback, context inspection, map display, and so on. Developers can exploit these tools to rapidly produce useful presentation clients. A key enabler for this integration is an object management facility. As a client-side layer, the Object Management Layer (OML) is capable of abstracting complexities associated with distributed object access, as well as offering a means for discovering various properties of the Context Tier.

The abstraction capability is particularly useful for clients that wish to *see* context as string-based symbols. GUI clients typically fall into this category as well as some symbolic inference engines. The discovery capability is the beginning of our intent for TIRAC to eventually support complete context discovery at runtime. Using discoverable context element templates, this facility currently supports the structural discovery of context in terms of element attributes and other structural properties. However, understanding that structure is only one part of true context discovery. OML to date does not attempt to support any type of semantic discovery.[2]

The second area of TIRAC's support for the *Presentation Tier* addresses presentation to software-based users, or external systems. The key issue in such support is the potential differences in representation between systems. In other words, similar to a GUI offering a graphical representation of the context to human users, presenting to a software-based user may also require some level of translation to a representation more native to the receiver. As discussed previously in Section 2.1.3, TIRAC addresses this need through its Translator facility, which is essentially a type of transformation engine that can be used to construct architectural

[2] Standardized exchange languages such as XML are also limited to conveying structure and not semantics. While structure can convey the organization of a chapter in a book, it does not convey what a chapter *is* or *implies*. It is generally well understood that semantic discovery offers a more significant challenge and is therefore currently the subject of considerable research.

bridges between systems operating over differing representations. Bridging, in this sense, not only supports functional connections but also connections at the representational level, a level critical in context-centric systems where much of the ability to interoperate with external systems depends on the richness and clarity of information or data feeds.

5.2.1 A Two-Layer Architecture

The shipment tracking and security application domain essentially calls for the seamless merging of an intelligent information management facility with existing data sources. As a TIRAC-based system this has been accomplished in SecureOrigins with an *information-centric* architecture that consists of two integrated components: a data-centric Data Capture and Integration Layer that incorporates linkages to existing data sources; and, an Intelligent Information Management Layer that resides on top of the data layer and utilizes collaborative software agents with automatic reasoning capabilities, serving as decision-support tools (Fig.33).

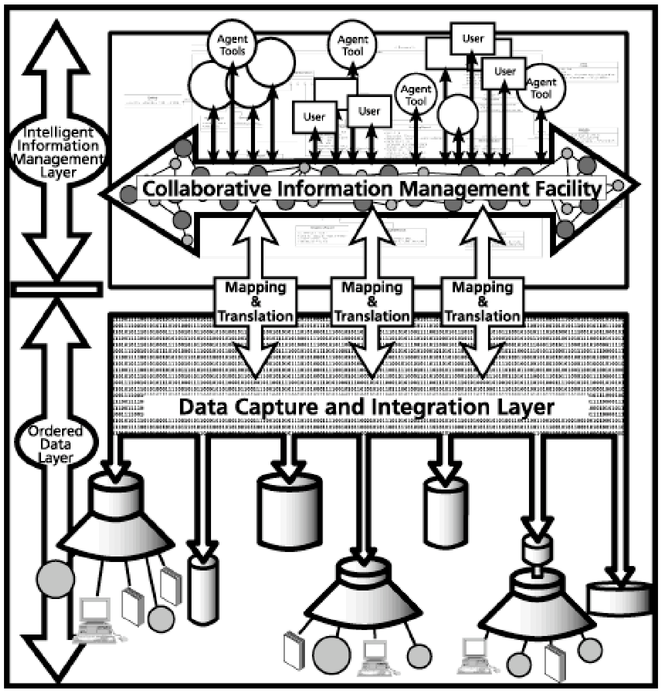


Fig.33: Schematic diagram of SecureOrigins™ system architecture

Data Capture and Integration Layer: The bottom layer of the SecureOrigins system takes the form of an operational data store and/or Data Warehouse, implemented within a commercial off-the-shelf relational database management system (RDBMS). This data categorization facility integrates data extracted on a periodic basis from several external sources into a *goods shipment* specific data schema. The design of the data schema is closely modeled on the structure of the ontology of the Intelligent Information Management Layer to minimize the required data-to-information and information-to-data mappings between the two system layers.

In conformity with normal Data Warehouse design practices the SecureOrigins Data Capture and Integration Layer incorporates the following four characteristics:

- It is subject-oriented to the specific business processes and data domains relevant to the shipment of goods across US borders.

- It is integrated so that it can relate data from multiple domains as it serves the data needs of the analysis functions performed by collaborative agents in the Intelligent Information Management Layer.
- It is periodically updated through its linkage to external data sources.
- It is time-based to support the performance of analyses over time, for the discovery of patterns and trends.

A multi-tier architecture is used to logically separate the necessary components of the data layer into levels. The first tier is the RDBMS, which ensures the persistence of the data level and provides the necessary search capabilities. The second tier is the service level, which provides the interface to the data level and at the same time supports the data access requests that pass through the mapping interface from the Intelligent Information Management Layer to the Data Capture and Integration Layer. It is designed to support request, response, subscribe, and publish functionality. The third tier is the control level, which routes information layer and user requests to the service level for the update, storage and retrieval of data. Finally, a view layer representing the fourth tier serves as a user-interface for the Data Capture and Integration Layer.

Intelligent Information Management Layer: The Intelligent Information Management Layer of the SecureOrigins system is designed as a typical TIRAC-based decision-support application. The core element of this application is an ontological framework to provide a relationship-rich model of the various system and application domains in which the application is required to operate.

5.2.2 Structure of the SecureOrigins Ontology

The underlying ontology of the SecureOrigins Intelligent Information Management Layer is divided into several somewhat related domains (Fig.34). While some of these domains describe application-specific events and information (e.g., goods movement transactions, shipping routes, and so on) others describe more general, abstract notions (e.g., event, threat, view, agent). The goal in developing the ontology was to abstract general, cross-domain notions into high-level domain models. As such, these descriptions can be applied across several application sub-domains. More domain-specific, concrete notions can then be described as extensions of these high level models.

Accordingly, the ontology of the Intelligent Information Management Layer has been modeled to include several primary meta-characteristics. Through inheritance, these meta-characteristics are propagated to extended, more specific ontological components. One of these meta-characteristics is the property of being *trackable*. This characteristic has been introduced at the ‘physical.Mobile’ level. Through inheritance, any entity that is a kind of ‘physical.Mobile’ automatically receives the property of being *trackable*.

A second meta-characteristic relates to the dispensability of an item. This property is represented at the ‘physical.item.Item’ level. Similar to the *trackable* characteristic, anything that is a kind of ‘physical.item.Item’ automatically receives the quality of being dispersible or *suppliable*. In addition, as an extension of ‘physical.Mobile’ such *suppliable* items are also *trackable*. Together, these two meta-characteristics provide an effective foundation for assigning basic logistical and tactical functionality to the application-specific domains identified within the ontology.

While meta-characteristics are only implicitly represented in an ontology, other additional notions may be represented explicitly in the form of object classes. For example, two such notions in the SecureOrigins ontology are *Container* and *Empowerable*. A *Container* holds (i.e., contains) components. In fact, a *Container* can be thought of as a dynamic set of components. There is no inherent order imposed. However, while the contents of *Containers* may be very different all *Container* objects have a common derivation. An *Empowerable* object is one that can be dynamically embodied, or enhanced with additional information, knowledge, or capabilities. This is intentionally a very open-ended notion allowing for unconstrained exploration into various potential *powers* that can be imparted to an object.

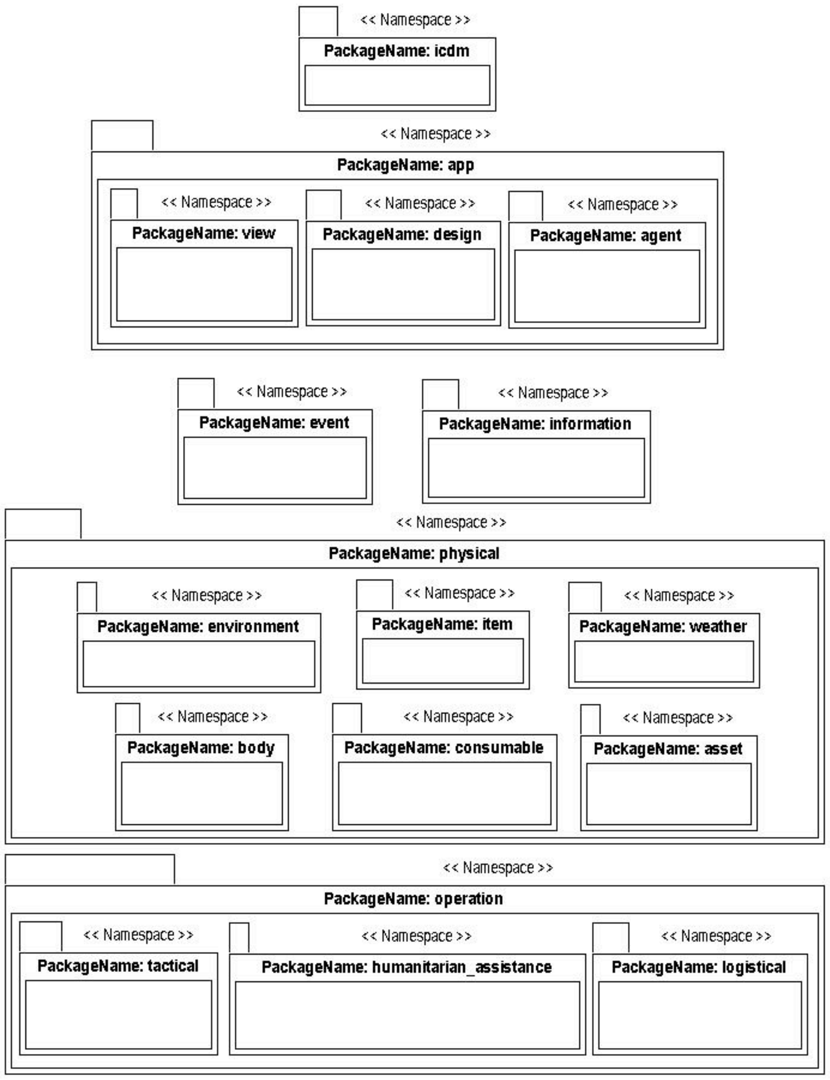


Fig.34: Typical ontology domains

Within the framework provided by the implicit meta-characteristics and explicitly represented notions, the SecureOrigins ontology includes a number of domains, such as:

Agent	Software module that has at least the ability to reason and communicate through an expressive language. In addition, *Agents* may have one or more of the following capabilities: maintain knowledge; act on their own initiative; collaborate with other agents to accomplish goals; and, use local information to manage local resources.
Application	Application software components and their relationships.
Shipment	Goods and products that are intended to be transported in some manner from a foreign country into the US. A *Shipment* is *suppliable* and *trackable*.
Body	Organization or people. A *Body* is *suppliable* and *trackable*.
Consumable	Through inheritance from *Item*, a consumable is *suppliable* and *trackable*. As the name suggests, a consumable can also be depleted.
Design	Describes a potential solution to a particular problem. Such a solution may identify various actions and processes needed for the realization of the solution. In general a *Design* also supports a set of requirements.
Environment	The natural (e.g., oceans, rivers, mountains, etc.) and artificial (e.g., ports, buildings, etc.) surroundings.
Information	Reports, manifests, bank statements, letters of credit, or any knowledge communicated or received concerning a particular fact or circumstance.
Item	A separate article. Being a kind of 'physical.Mobile' an *Item* also has the property of being dispersible. That is, an *Item*, along with any of its derivations, is not only *trackable* but also *suppliable*.
Mobile	Any physical entity that moves or can be moved. As such, a *Mobile*, and any of its derivations are *trackable*.
Operation	A set of tasks or processes designed to achieve one or more objectives.
Physical	Anything pertaining to materials, whether natural, man-made, or organic.
View	A particular version of reality. Reality itself can be considered a *View*. Existing as a *Container* a *View* contains all information, entities, events, and so on. needed to describe itself. *Views* may be shared among any number of users. *View* components may be copied between *Views*. Also, *Views* may be merged with other *views* employing various forms of conflict identification and consequential resolution to deal with the inconsistencies that may arise.

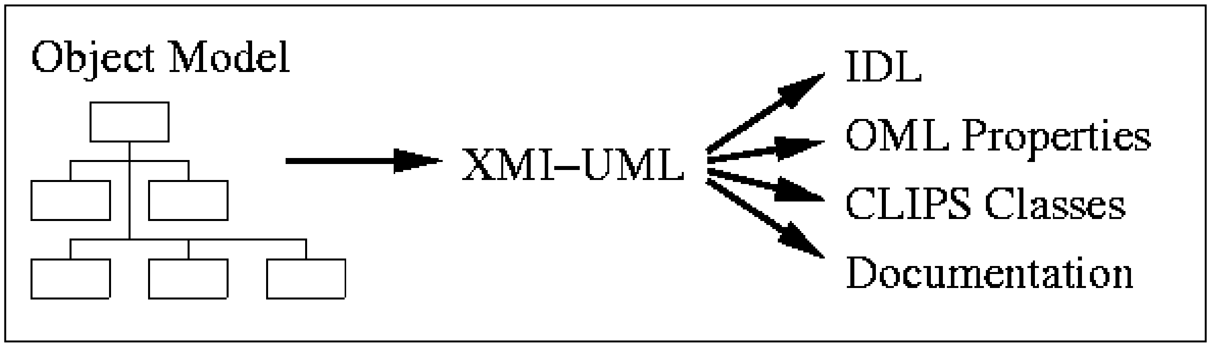
5.2.3 Model Maintenance

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 TIRAC-based applications. In addition, SecureOrigins utilizes the Object Constraint Language (OCL) for the definition of specialized presentation mappings.

5.2.4 Artifact Generation

In the TIRAC software development environment all artifacts are generated from XMI-UML in any of the following formats:

1. CORBA Interface Definition Language (IDL)
2. Object Management Layer (OML) Class Properties
3. CLIPS Class Definitions
4. Model Documentation



5.2.5 The Object Management Layer (OML)

As a TIRAC-based application SecureOrigins takes advantage of the object management facilities provided by the OML component of TIRAC. 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. Interests may be constrained through the use of conditions. Condition satisfaction checking is performed utilizing an inference engine. 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.

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.

5.2.9 SecureOrigins Agent Engine Example

The particular Agent Engine configuration that was developed for the TIRAC-based SecureOrigins application, monitors and reacts to changes in the characteristics and relationships of the information expected during the transportation of goods from their originating location, through multiple checkpoints including an international border (e.g., the US-Mexico border), to their final destination. Depending on the conditions encountered within this goods shipment environment, the SecureOrigins Agent Engine 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 that aid the operator in the decision making process. Alerts are displayed through the Agent Status Panel, a component that can be integrated into the SecureOrigins Graphical User Interface or operate independently. It is through the Agent Status Panel that the operator interacts with the Agent Engine.

The issue of conflict resolution in such a highly collaborative agent-based environment requires further elaboration. Simply stated, the problem involves the coercion of groups of collaborating agents to deal with inevitable *differences of opinion* and constructively converge on a particular decision or solution. This is especially prominent in opportunistic collaborative models where collections of agents offer their unique expertise-based perspectives in the formulation of some type of design or plan.

Consider an agent taxonomy identifying three basic types of agents. These agent types include domain agents, object agents, and mediation agents (Pohl 1996, 1998). Domain agents are essentially service-oriented agents with each subtype embodying expertise in a particular application-relevant domain (i.e., structural systems and thermal dynamics for architectural design, tidal dynamics and trim and stability for ship load-planning, payload tampering vulnerability for transportation security, etc.). The collection of domain agents operating on a given problem at any point in time determines the diversity of domain specific perspectives and the analytical depth involved in inter-agent discussions.

Object Agents, on-the-other-hand extend the notion of high-level contextual representation by essentially *agentifying* key entities within a scenario through empowering their context model representation with the ability to not only initiate action but to do so in their own interests. In this sense, each object agent collaborates within agent-level discussions with the biases associated with its own interests. Interests can be based on a variety of objectives including survival, enrichment, or even domination. The particular value model to employ depends on the nature of collaboration desired (e.g., sustainment, competitive, etc.). Depending on the collaborative model employed the ability and likelihood for communities of collaborating agents to come to agreement can be somewhat remote. This is where the third type of agent, the mediation agent, comes into play.

In an attempt to resolve conflicts or impasses arising between collaborating agents, mediation agents play the role of skilled third party mediators. Mediation agents are equipped with an objective to resolve collaborative conflicts along with expertise and, if embodied with the ability to modify their logic based on performance, with experience in the domain of *mediation*. Mediation agents monitor collaboration among agents looking for signs of non-convergence. Once detected, a mediation agent progressively applies an arsenal of well-documented strategies in an attempt to find a workable decision and allow discussions to continue.

Like the flexibility afforded in managing alternative sets of related context, TIRAC's support for the Logic Tier refrains from constraining development to one particular collaborative model or paradigm. Rather, TIRAC offers a set of fundamental capabilities upon which a variety of approaches can be applied.

Within the TIRAC Agent Engine there exist two types of agents. Static service agents are created at the beginning of an Agent Session and generally monitor the problem space represented by the ontology, 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 the application. Dynamic mentor agents are not necessarily created at the beginning of an Agent Session, rather their creation may be dependent on the occurrence of a particular event or the creation of a specific object instance type. For example, although not yet implemented, it is envisioned that the SecureOrigins application will eventually also feature mentor agents that are automatically created as a truck leaves the plant on the Mexican side of the border. It would be the responsibility of such a mentor agent to track and monitor the truck in respect to specific security parameters until it reaches its destination on the US side of the border. Once their mission has been completed, mentor agents are usually automatically terminated.

In its initial incarnation the SecureOrigins Agent Engine was designed to include five agents, namely: a *Route Agent* that tracks the progress of shipments through checkpoints with the objective of detecting time delays and route deviations; a *Weather Agent* that monitors local weather conditions; a *Security Agent* that verifies and monitors the authenticity of the driver, the condition of the conveyance and the sensors that are installed as a security shield around the goods loaded onto the conveyance; a *Threat Agent* that monitors hazardous cargo and assesses local crime and national threat conditions; and, a *Transport Agent* that determines the implications of multiple agent warnings and alerts. These agents will automatically analyze the particular data pertaining to each goods shipment, share their findings with each other, and react to changes in information as they monitor each shipment from point of origin to final destination. Typical warnings and alerts include:

(a) Warning that hazardous material is en route.

(b) Warning that a truck has not reached a waypoint within time limit 'A'.

(c) Alert that a truck has not reached a waypoint within time limit 'B'.

(d) Warning that a truck is near a higher risk (e.g., crime) area.

(e) Alert that a truck has stopped for more than 'C' minutes near a higher risk area.

(f) Warning that a replacement driver of low risk is driving a truck.

(g) Alert that a replacement driver of elevated risk is driving a truck.

(h) Alert that the loaded weight of truck does not match final weight at the border.

(i) Warnings and alerts that a particular combination of circumstances involving encyclopedic data and truck-based/convoy-based confirmation data entered at waypoints and checkpoints constitutes a higher risk situation (e.g., a particular driver transporting certain kinds of goods, and so on).

(j) Warnings and alerts that a particular combination of circumstances involving encyclopedic data, truck-based/convoy-based confirmation data entered at waypoints and checkpoints, and periodically updated context information, constitutes a higher risk situation (e.g., a particular route under certain weather conditions, and so on).

Provision has been made in the SecureOrigins Agent Engine to take future advantage of anticipated advances in automated image recognition capabilities. This will allow the establishment of linkages between the reasoning capabilities of the software agents and the visual surveillance of the loading, movement and unloading of conveyances, as well as the visual monitoring of sites of interest such as parking areas, streets, warehouses, and loading docks. For example:

(k) Warning or alert with direct reference to a particular camera. Eventually, this could obviate the need for human operators to continuously monitor the images displayed by video cameras.

(l) Warning or alert with direct reference to a specific area of the visual field displayed by any particular video camera.

(m) Automatic manipulation of a particular camera (e.g., moving the visual field or zooming in on a particular area of the current visual field) based on an agent warning or alert.

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.35. 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’.

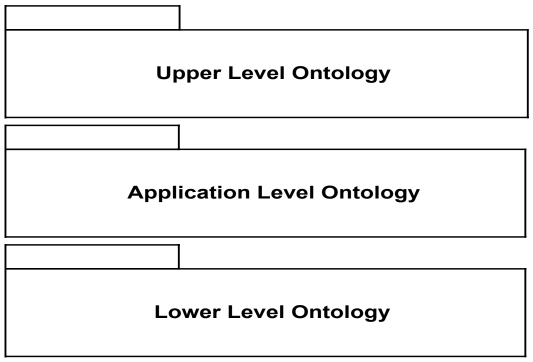


Fig.35: Ontology semantic levels

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. Furthermore, 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.36).

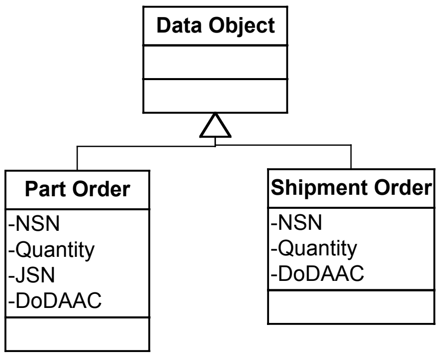


Fig.36: Data Level Interface specification

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.36) 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.

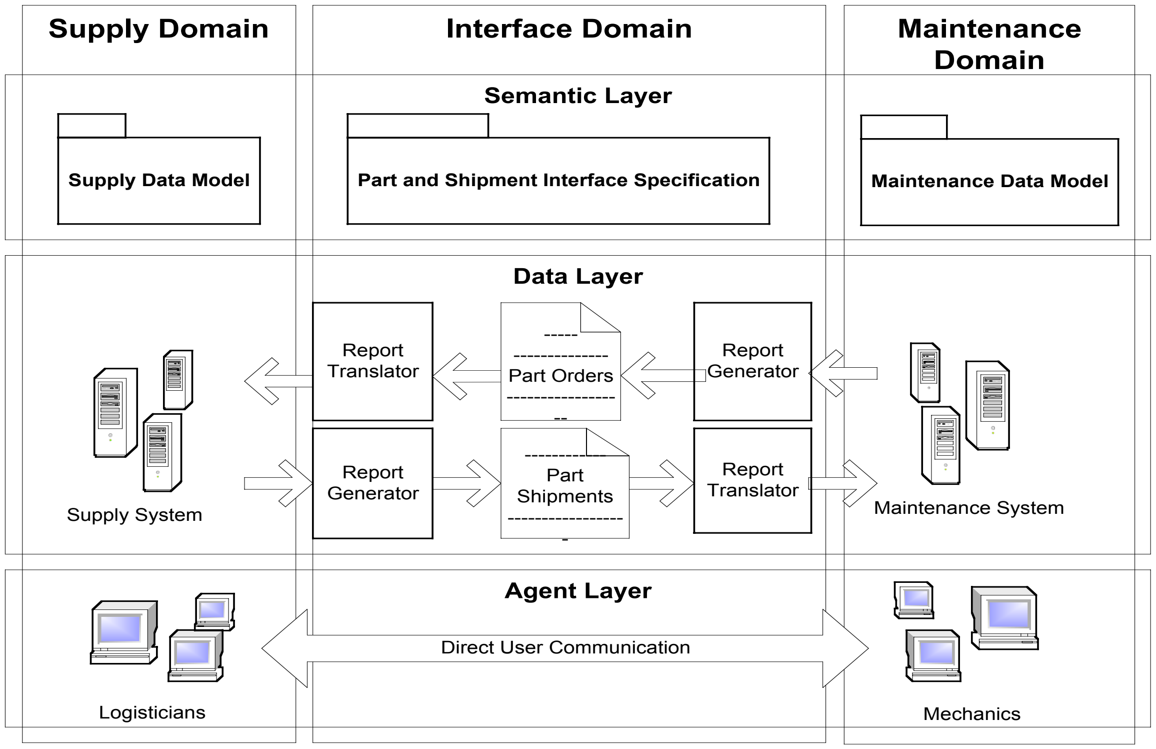


Fig.37: Data Level System Interface

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.37. 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 parts 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 parts 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 in Fig.37 are representative of these hidden context shifts into hidden proprietary data models.

As data level system interfaces such as those 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.38 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.

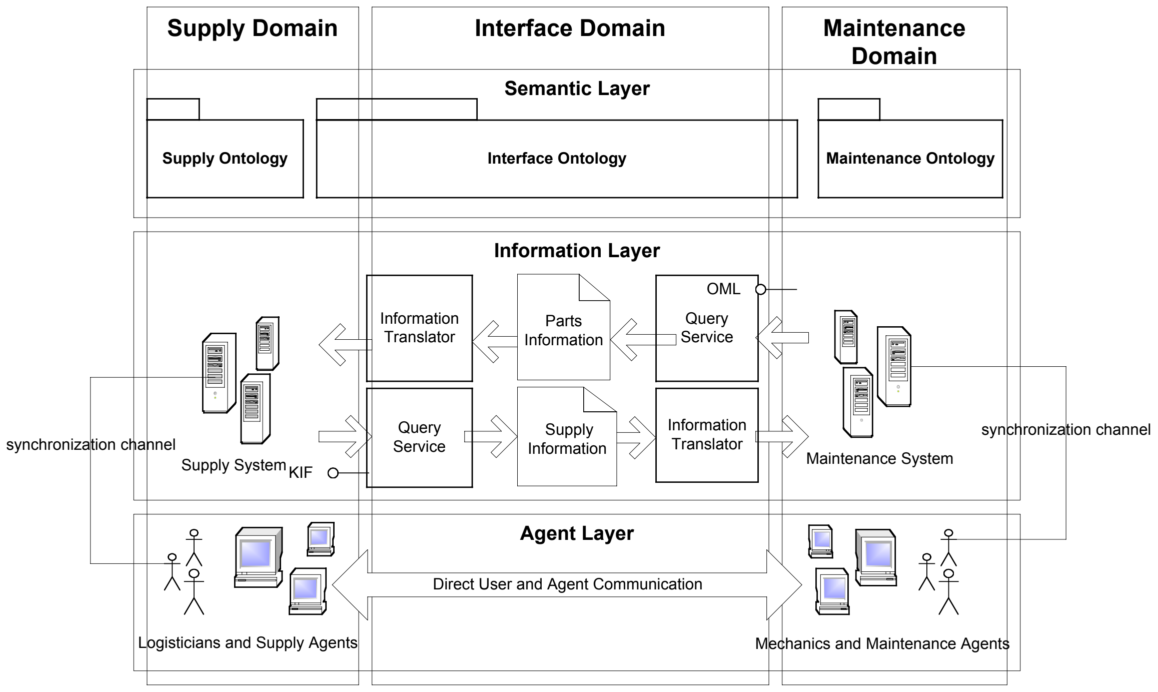


Fig.38: Information Level System Interface

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 TIRAC-based Object Management Layer (OML) employed by most of the non-military systems developed by CDM Technologies.

Between the information and agent layers in Fig.38 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.

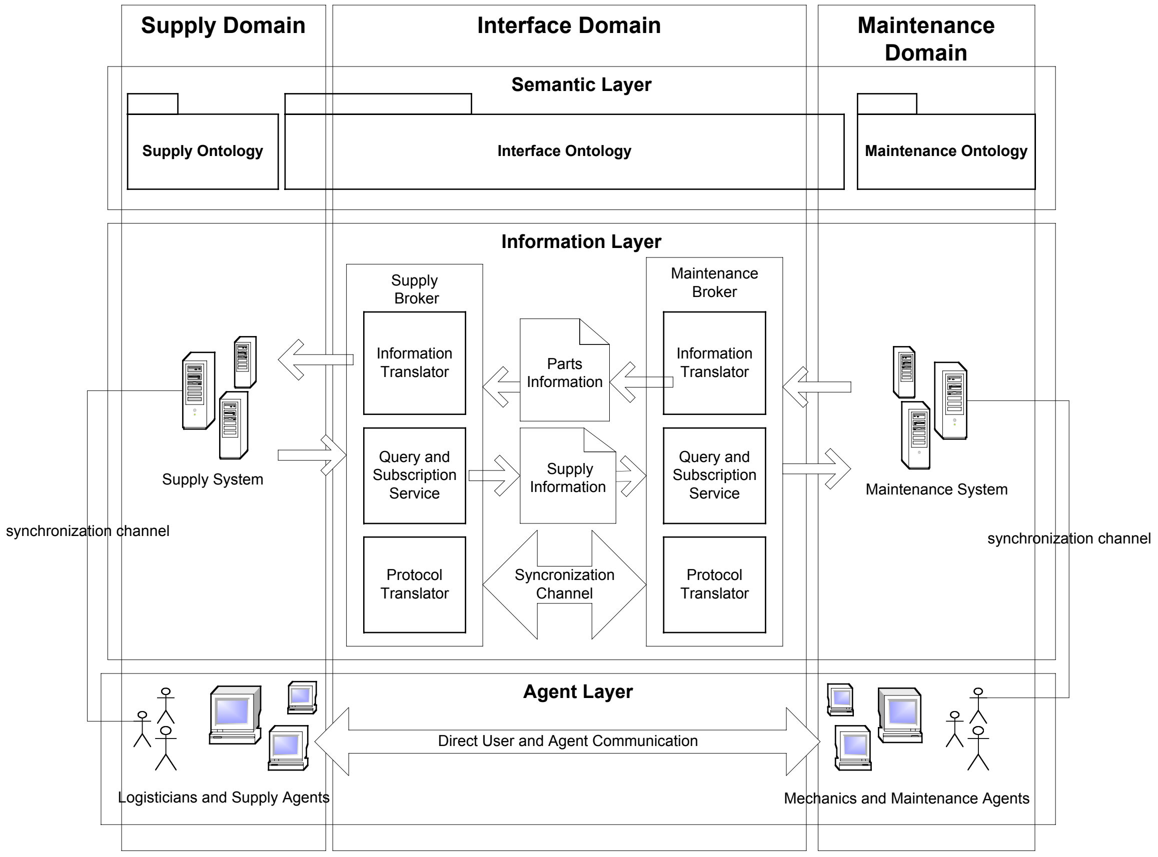


Fig.39: Information level system interoperability

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.39. 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 homeland security domain, one could envision the proposed system of systems hierarchy following along the lines of existing unit hierarchies within the federal, state and local government services down to the first responders, with the top level of the hierarchy operating at the level of the Secretary of the Department of Homeland Security and ultimately the President. 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.40). 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.

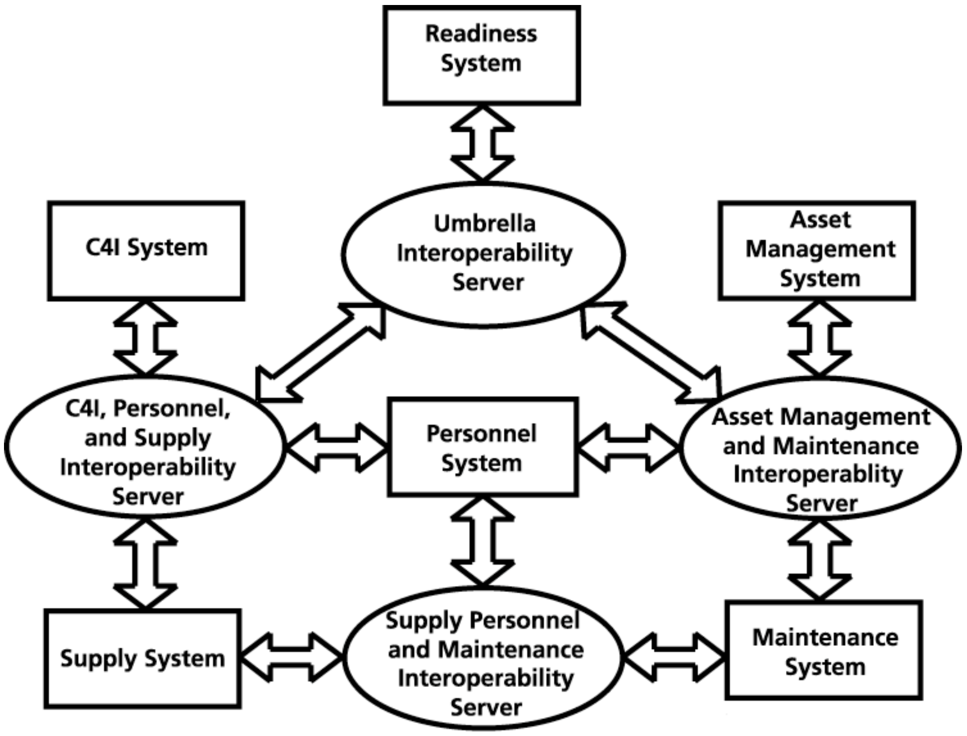

Readiness System
Umbrella Interoperability Server
C4I System
Asset Management System
C4I, Personnel, and Supply Interoperability Server
Personnel System
Asset Management and Maintenance Interoperablity Server
Supply System
Supply Personnel and Maintenance Interoperability Server
Maintenance System

Fig.40: Interoperability Server

For example, the Federation Interface Ontology for a multi-modal 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, 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 goods movement or shipping system an entity such as a truck may be viewed, and therefore represented as a type of conveyance. In this case the bias would be toward shipping utility. However, in a maintenance system the same truck would most appropriately be viewed as an object of repair with a corresponding emphasis on spare parts and labor. In both cases, however, the subject is still the exact same truck 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 planning of inventory control and product distribution then it is more appropriate and beneficial for an entity such as a electric power generator to be primarily viewed as a shipping unit rather than as an electricity producer. If viewed as a shipping unit the description of the generator 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 generator. In the context of its intended use, however, such information is of much less (if any) interest. Information more useful to a potential customer would include technical performance and cost characteristics such as capacity, life-span, maintenance requirements, etc. Conversely, these characteristics are of little or no significance in the domain of storage and shipping. Nevertheless, regardless of the perspective it may be the exact same electric power generator that is being discussed between the two disparate systems. 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 wishes 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.41 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.

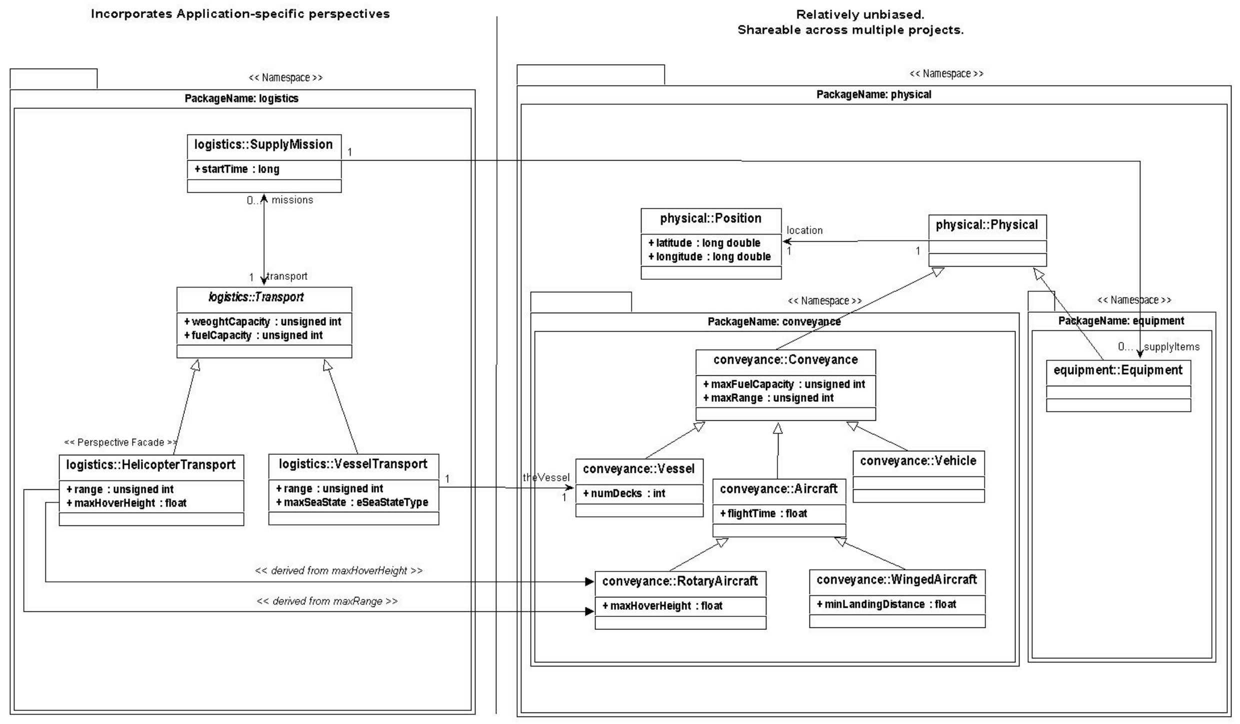


Fig.41: Partially derived logistics ontology

Core of the logistics perspective presented in Fig.41 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 system while still employing the core conveyance ontology. As Fig.41 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.41 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.41 is the notion of a *SupplyMission*. Being a fundamental concept in the example system a supply mission (i.e., customer order fulfillment) essentially relates supply items in the form of products to the transports by which they will be delivered. Once again, the definition of a shipment-specific notion (i.e., supply items) is derived from a notion defined in the core ontology (i.e., product or equipment). In this case, an explicit relationship is declared linking *SupplyMission* to zero or more *Equipment* items. Since, from the perspective of the shipment 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.42 briefly extends the inventory control and shipment example presented in Fig.41 showing collaboration between the two systems. 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 regardless of how it is viewed in each system. To represent domain-specific notions 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.

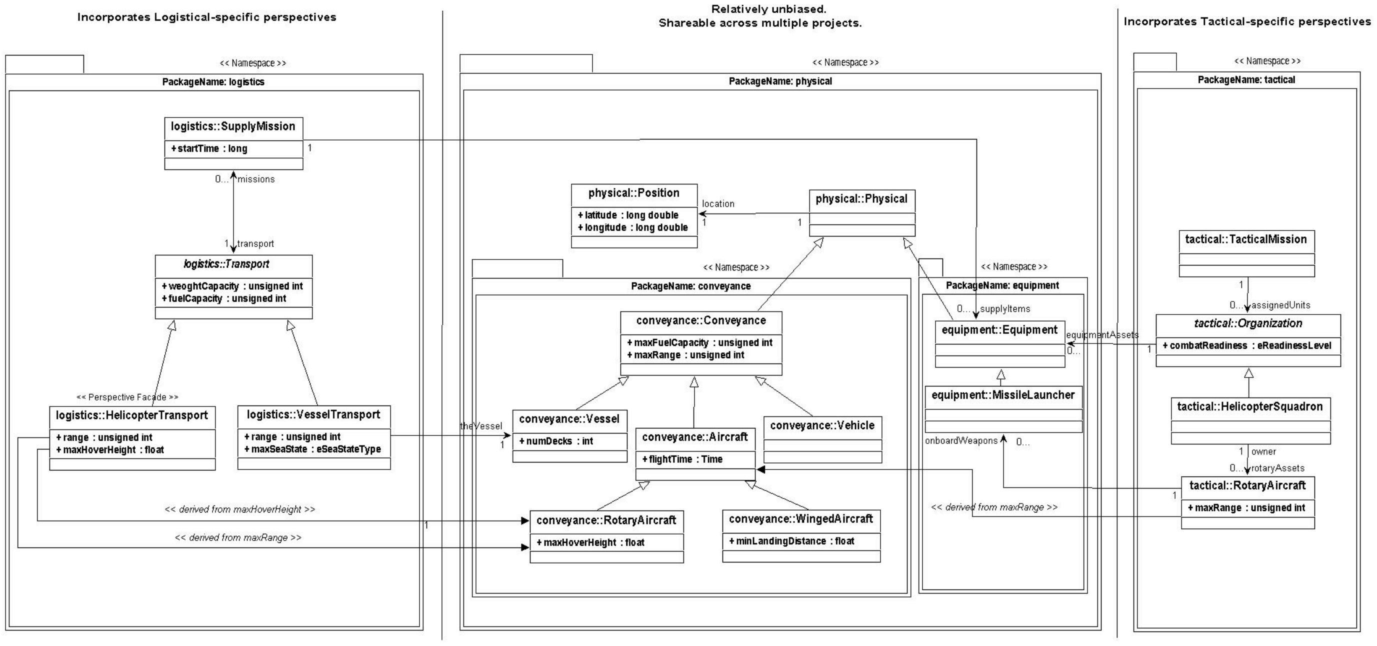


Fig.42: Two disparate domains linked into the same core ontology

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 TIRAC-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.

Brickley D. and R. Guha (eds.) (2002); 'RDF Vocabulary Description Language 1.0: RDF Schema'; W3C Working Draft, April 30. (www.w3.org/TR/rdf-schema/)

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.

Cohen P, R. Schrag, E. Jones, A. Pease, A. Lin, B. Starr, D. Easter, D. Gunning and M. Burke (1998); 'The DARPA High Performance Knowledge Bases Project'; Artificial Intelligence Magazine, 19(4), (pp. 25-49).

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

Daconta M., L. Obrst and K. Smith (2003); 'The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management'; Wiley, Indianapolis, Indiana.

Dean M., D. Connelly, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, F. Stein and L. Stein (2002); 'OWL Web Ontology Language 1.0 Reference'; W3C Working Draft 29, December 12 (www.w3.org/TR/owl-ref/)

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.

Freidman-Hill, E. (2003); 'Jess In Action'; Manning Publications, Greenwich, Kentucky.

Gil Y. and V. Ratnakar (2002); 'Markup Languages: Comparison and Examples'; University of Southern California (USC), Information Sciences Institute, TRELLIS Project (www.isi.edu/expect/web/semanticweb/comparison.html)

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

Goodman S. and J. Pohl (2003); ‘‘ICODES: A Ship Load-Planning and Control System’; 13th Ship Control Systems Symposium (SCSS), Orlando, Florida, April 7-9.

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

Holguin H. (2003); ‘SecureOrigins: National Strength Through Security and Competitiveness’; Office of Naval Research Workshop on ‘Developing the New Infostructure’, Proceedings, CAD Research Center, Cal Poly, San Luis Obispo, California, September 10-11.

Horrocks I. (2002); ‘DAML+OIL: A Description Logic for the Semantic Web’; IEEE Intelligent Systems, Trends and Cotroversies.

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

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).

Pohl K. (1998); 'The Round-Table Model: A Web-Oriented Agent-Based Framework for Decision-Support Applications'; in Pohl J. (ed.) Advances in Collaborative Decision-Support Systems for Design, Planning, and Execution, focus symposium: International Conference on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, August 17-21, (pp.47-59).

Quartel, R. (2002); Testimony before the Senate Committee on the Judiciary Subcommittee on Technology, Terrorism and Government Information, on 'Securing Our Ports Against Terror: Technology, Resources and Homeland Defense'; February 26.

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

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.

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

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

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.

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

abstraction 31-32, 64
accessibility 30
accounting 10
actors 69
adaptor 26
adaptability 29, 32-34, 41
Agent Engine 24, 28, **51-53**
Agent Manager 27
Agent Status Panel 52
agents 1, 7, 13-14, 16-18, 21-24, 29, 34, 42-44, 46-47, 52-55, 61, 71
  alerts 21, 45, 51, 53-54
  community 25
  domain agents 52
  Facilitator Agent 15
  mediator 52
  Mentor Agent 14-16, 22, 52-53
  Object Agents 52
  Planning Agent 15
  Route Agent 53
  Security Agent 17, 53
  service agents 15, 52
  status 25
  Threat Agent 53
  Transport Agent 53
  Trim and Stability Agent 15-16
  Weather Agent 53
Agent Session 52
AI (Artificial Intelligence) 11
alerts 21, 45, 51, 53-54
algorithms 6, 32
ALO (Application Level Ontology) 56
API (Application Programming Interface) 30, 51
architecture 22-24, 26, 29, 38-39, 46-47, 52, 70
ASCII (American Standard Code for Information Interchange) 13
associations 2, 12, 14, 50
attributes 14, 18, 26-27, 32, 50, 58, 66-67
autonomous 40
Ayers 38

B

bank statement 43
bar code 42
behavior 30, 31
BEL (Business Engine Layer) 34
bias 64
Bill of Lading 44
binary 13
biological 41
bits 6-7
bookkeeping 9
border 16, 41-43, 47, 53
Border Check Point 42
Brickley 21
bridges 5, 22, 26, 33, 45
BRL (Business Rule Layer) 34
Bruce 11
Buschmann 65
business intelligence 9-10, 17-18

C

C++ 11
CADRC (Collaborative Agent Design Research Center) 6, 28-29, 35
Cal Poly (California Polytechnic State University, San Luis Obispo) 6, 28
calendar management 25, 45
camera 42
Canada 41
cargo 16
CDM Technologies Inc. 21, 28-29, 35, 41, 60
Certificate of Origin 34
Chandrasekaran 30
characteristics 7, 14, 47, 50
chemical 41-42
classes 50
CLIPS (C Language Integrated Production System) 37, 39
Cockburn 69
code generation 27-28, 34
Cohen 21
collaboration 6-7, 9, 14, 21-25, 29-32, 34, 38, 43-44, 47, 52, 64-65, 67
common language 14-17
competition 10, 18
computation 15, 16, 37

concepts 16, 21, 31, 64
conflict 15, 21, 52
connectors 26
constraints 63-64, 66
container 16, 41, 44
content 6
context 1, 6-7, 9, 11, 17, 21-26, 29-30, 34, 44-45, 52, 55, 57, 59, 63-64, 68
context-centric 23
context model 23
Context Tier 23-24, 27, 45
coordination 24
CORBA (Common Object Request Broker Architecture) 23, 38-39, 50
Cornell 11
cost 44
coupling 32, 62
crane 16
crisis coordination 21

D

Daconta 22
DAML (DARPA Markup Language) 21
DARPA (Defense Advanced Research Projects Agency) 21
data 6-7, 11, 15, 18, 21-23, 55, 57, 68
Data Capture and Integration Layer 46-47
data-centric 5-6, **9**, 13, 17, 45-46, 57, 59
Data Mart 10, 18
data-processing 10, 11, 17
Data Warehouse 10, 46, 71
Daylight 6
DBMS (database management system) 10, 42, 46
DDDS (Defense Data Dictionary System) 58
Dean 21
decision-support 7, 9, 15, 17, 21-24, 29, 32, 34-35, 37-48, 40, 44, 46, 64-65, 68
deck 16
definitions 7
Denis 11
Department of Homeland Security 62
depreciation tables 9
design 29, 52
digital 13
disaster 24
discovery 22, 24 26, 33, 45
distributed 1, 9, 21, 23, 25, 31, 40, 45
DLSI (Data Level System Interface) 58-59

DoDAAC (Department of Defense Activity Address Code) 58
domains 48, 55-56, 65
downsizing 10

E
e-commerce 11
EVP Group 41
El Paso 41
elevator 16
e-mail 13, 31
EMMERS 21
encapsulation 32
encyclopedic data 53
end-user 55, 68-69, 71
engineering design 6
evaluation 21
event 22-23, 25, 39, 47, 51-52
exchange languages 26
execution 2, 30
expert system 21, 40
express lane 44
extensibility 32, 34

F
façade 2, 17, 31-33, 65, 67
filter 64-68
financial transactions 43
first responder 24, 62
flexibility 29, 32, 52
Forgy 38
Fowler 65

G
gaming 2
gamma rays 42
Gil 21
Ginsberg 11
global reference 30
Goodman 16
goods shipment 16, 21, 41-43, 46, 51, 53
GPS (Global Positioning System) 16, 42

GUIL (Graphical User Interface Layer) 34
Gray 23, 37
guardian angel 14
Gross Domestic Product 41

H

hazardous material 16, 53
heuristics 11
Holguin 16, 35
Holguin Group 41
homeland security 41, 62
Horrocks 21
Horstmann 11
Humphries 10

I

ICADS (Intelligent Computer-Aided Design System) 35
ICODES (Integrated Computerized Deployment System) 16, 17
IDL (Interface Definition Language) 50
image recognition 53
incentive 44
indemnification 44
indexing procedures 11
indirection 67
inference engine 24, 27, 37, 40, 50
information 6-7, 9, 11, 14-15, 17-18, 21-23, 33, 37, 55, 68
information-centric 1, 5, 7, 15-18, 22, 30, 32, 44-45, 59, 62, 64-65, 67, 70
Information Level Interface **59-61**
Information Level Interoperability **61-64**
Information Management Layer 46-47, 70
information-serving collaboration facility 17
Information Server 23, 28
inheritance 11-12, 47
inspection 41-42, 44
interest rate tables 9
interface brokers 61
Interface Domain 9
Interface Domain Ontology 55
Internet 11, 55
interoperability 5, 16-17, 55-56, 61-65, 68
interoperability servers **62-64**
interpretation 9, 11, 13, 55
interrelationships 7

in-transit visibility 15-16, 44

J
Java 11, 50
Jess 24
JSN (Job Serial Number) 58
Juarez 41-42

K
Kleppe 21
knowledge 7, 9, 14, 17-18, 22-23, 55
Knowledge Interchange Format (KIF) 60
KOALA (Knowledge-Based Object-Agent Collaboration) 35

L
language 7
licensing 27, 28
Letter of Instructions 44
Lievano 23. 37
lighting 30
load balancing 2
load-plan 16, 52
logic 23, 33, 37
Logic Tier 34, 52
Lower Level Ontology 56

M
Mallon 42
maintenance 56, 58
meaning 6-7, 21
Mexico 41-42, 51, 53
modeling 2
monitoring 21
motion detection 42
Mowbray 34
multi-layered 29, 33-34
multi-tiered 29, 33-34, 47

N

NASA 37
natural language 11
Navy 56, 58
neutron rays 42
notions 16, 21, 31
NSN (National Stock Number) 58
nuclear 41
Null Server 51

O

OAL (Object Access Layer) 34
Object Environment Server 51
Object Manager 25
object-oriented 11, 57
Object Server 23, 51
object 2, 7, 11-16, 18, 22, 27, 31-33, 38-39, 48, 50, 55
  model 11-12
  shared 31, 38
  wrappers 33
OCL (Object Constraint Language) 50
OIL (Ontology Inference Layer) 21
OLAP On-Line Analytical Processing) tools 10
OML Object Management Layer) 32, 34, 45, **50-51**, 60
ontology 1, 7, 13-18, 21-23, 27, 30-34, 37-39, 45-46, **47-48**, 52, 55, 58, 59, 62, 64-66, 68-69
  Application Level Ontology 56
  Interface Domain Ontology 55
  library 68
  universal 65
open architecture 24
Operational Data Store 71
opportunistic 34, 52
OTL (Object Transport Layer) 34
OWL (Web Ontology Language) 21

P

payroll 10
Pedersen 11
performance 24
perspective 31, 64-65

PLANMAX 21
planning 2, 21
playback 25, 45
Pohl J. 15, 16, 28, 29, 35, 52
Pohl K. 15, 29, 31, 35
predicate logic 37
presentation 22-23, 33, 37, 45
Presentation Tier 25, 34, 37, 45
priority 70-71
process 27, **55-71, 68-71**
profile 23-24
profiling **43-44**
property file 34
proprietorship 27-28
purchasing documents 43

Q

Quartel 43
query 16, 23, 30, 37, 55

R

radiation 42
random 44
Ratnakar 21
RDF (Resource Definition Framework) 21
real-time 38, 45
reasoning 2, 6. 14, 21, 30, 31, 34, 46, 64
record storage 9
Referential Integrity 50
reflection 50
rehearsal 2
relationships 7, 11-12, 14, 18, 21, 27, 50, 57, 64, 66
reliability 44
remembering 14
reports 9, 25, 45, 59
representation 5, 7, 11, 14, 16, 22, 24, 26, 31, 34, 52, 56, 65
Rete 37
reuse 32
RF-Tags 42
risk 42, 45, 53
RMI (Remote Method Invocation) 23
room 6

routes 47, 53
Rule Generator 39
rules 37, 39, 40

S

search 23
security 16, 21, 41-43, 45, 52
SecureOrigins 16, 17, 21, **41-54**
self-association 2
self-diagnosis 2
self-healing 2
self-monitoring 2
Semantic Network 27
Semantic Network Manager 27
semantics 21, 26, 57, 64
services 15, 22-23, 27, 30, 31
  subscription 15, 23
ship 17, 21
shipper 41-43, 47, 64
smuggling 41
SOAP (Simple Object Access Protocol) 22
software 55-71
software process **55-71, 68-71**
spaces 30
SQL (Structured Query Language) 26, 60
statistics 11
storage 5, 14
stove-piped 5
Stroustrup 11
structural systems 52
subscription 1, 15-16, 23, 30-31, 37-40, 61
  profile 1, 15-16, 30-31
  service 16, 23, 30, **37-40,** 61
Subscription Server 37, 64

T

taxonomy 11, 13
telephone 18
terrorists 16, 41, 43
Texas 41
thermal dynamics 52
thieves 16

threat 47
tiers 23
time 30
tools 23, 27, 29, 41
transformation 22, 26, 45
translator 17, 26-27, 45
trim and stability 15, 16, 17, 52
trusted source 42

U

UDDI (Universal Descriptor Discovery and Integration) 22
UML (Uniform Modeling Language) 21, 27, 50
understanding 14
US (United States) 41
universal ontology 65
use-cases **68-71**
user-interface 27-28, 45

V

variable 22
Verity 11
view 17, 22, **24**, 25, 47, 65
vocabulary 55, 58

W

walls 30
Warmer 21
warning 21, 45, 53-54
Web 22
window 6
work flow 10

X

XMI (XML Metadata Interchange) 50
XML (Extensible Markup Language) 21, 26-27, 45, 50
XSL (Extensible Stylesheet Language) 26
XSLT (Extensible Stylesheet Language Transformation) 21, 26

Y

Z

Zahavi 34