

U.S. Army Aviation and Missile Command
Acquisition Center
Aviation & Missile RDEC Directorate

A Feasibility Study using Intelligent Software Agent
Technology for Weapon-Target Pairing, utilizing
Unmanned Aerial Vehicles
Scientific and Technical Report

December 30, 2008

The EPV Group, LLC
9 Civic Center Plaza
El Paso, Texas 79901

In Collaboration with

CDM Technologies Inc.
San Luis Obispo, California

(This page intentionally left blank)

CDRL# DI-MISC-80711A: Scientific and Technical Report
Intelligent Software Agent Technology for Unmanned Aircraft Systems (UAS)
A Feasibility Study

Executive Summary

This report explores the feasibility of applying artificial intelligence (AI) technology and in particular intelligent software agent methodologies to facilitate the remote guidance and improve the autonomous control capabilities of unmanned aerial vehicles (UAV). In their primary military applications of surveillance, target identification and designation, counter mine warfare, and reconnaissance, the effective deployment of UAVs is currently limited by inadequate software support. The two most critical software needs are cited in the literature and government reports as a reliable autonomous capability to detect, assess and respond to near-field objects in the UAV's path of travel and the ability to process large volumes of data collected by multiple UAVs in near real-time.

The findings of this study indicate that both of these needs can be better addressed by AI-based information-centric software than legacy data-processing software systems. While AI-based software is not a magic wand that can today fulfill all of the control and data processing needs of unmanned aircraft systems (UAS), it has reached a sufficient level of maturity to warrant immediate exploratory implementation. It is important that the developers of AI-based software become aware of the needs and capabilities of UAS users and manufacturers, and that vice versa, the UAS community become familiar with AI concepts and implementation principles. Since both UAS and AI are relatively young technologies there is a need for the early teaming of members of these communities to guide the research and development efforts and thereby accelerate the effective integration of AI capabilities in UAS software systems.

The report is divided into eight main sections. Section 1 introduces intelligent software as a paradigm shift that is manifesting itself in the transition of data-processing to knowledge management and then states the purpose of this feasibility study. Section 2 briefly summarizes the primary military applications of UAS capabilities and lists the principal technical challenges identified in the literature. In particular, this Section provides an analysis of the *Unmanned Systems Roadmap 2007-2032* published in December 2007 by the Office of the Secretary of Defense (OSD 2007). While this publication mentions the need for and promise of AI technology, it generally underestimates in the opinion of the authors current capabilities and the current level of maturity of AI-based software methodologies. Specifically, it overlooks the need for UAS and AI subject matter experts to team together so that the needs of the UAS community can provide appropriate guidance to potentially useful AI research and development efforts.

Section 3 provides an introduction to selected AI concepts and software architecture design and implementation principles. An explanation of the difference between information-centric and data-centric software stresses the role that *context* plays in the automated interpretation of data. Such interpretation capabilities are a fundamental prerequisite for the timely and effective merging and management of the very voluminous data streams that are typically collected by UAVs on surveillance and reconnaissance missions. Following an explanation of service-oriented architecture

(SOA) design concepts and implementation principles, the purpose and functions of SOA components such as services management framework (SMF), enterprise service bus (ESB), and enterprise services are described in conceptual terms. Section 3 concludes by emphasizing the need for integrating AI methodologies with SOA principles to produce software systems that provide flexibility and interoperability in support of reusable, intelligent tools.

Sections 4, 5 and 6 explain the underlying notions and describe the capabilities of: various kinds of intelligent agents; Web services as an implementation of SOA concepts; and, interoperability bridges for facilitating the meaningful transfer of data. This Section builds on the AI technology underpinnings presented in Section 3 and shows how information-centric software principles become an enabler of intelligent tools and services within a SOA-based system environment. Emphasis is placed on the representation of *context* to enable the automated reasoning capabilities of rule-based agents for planning, re-planning, truth maintenance, and learning functions. However, intelligent agent implementations for UAV systems are likely to be of a hybrid nature involving a variety of AI methodologies including in addition to rule-based agents also neural networks, genetic algorithms, swarm intelligence, and case-based reasoning.

In Section 7 UAV image interpretation serves as the basis of an example application of AI technology to UAV object detection and tracking. A proof-of-concept demonstration system has been developed as part of this feasibility study, utilizing the on-line Adaboost algorithm developed by Grabner and Bischof (2006). By combining the results produced by several Weak Classifiers the Adaboost algorithm is able to dynamically create a single Strong Classifier capable of providing a significantly more reliable object detection result.

Section 8 describes a Phase II follow-on effort that would result in the development of a prototype Unmanned Vehicle Object Recognition and Interpretation System (UVORIS). The proposed capabilities of UVORIS are focused on object detection, tracking and identification, combining several AI methodologies within a SOA system environment. The technical approach includes case-based reasoning, ontology representation, rule-based agents, statistical object recognition, and semantic database search techniques. Section 8 concludes with task statements and estimated costs for a 10-month Phase II effort.

Intelligent Software Agent Technology for Unmanned Aircraft Systems (UAS) A Feasibility Study

Table of Contents

Executive Summary	1
Table of Contents	3
1. Introduction: <i>Intelligent Software as a Paradigm Shift</i>	5
1.1 Purpose and Structure of Report	6
2. Background: <i>Unmanned Aircraft Systems</i>	7
2.1 Definitions	7
2.2 Primary Military Applications	8
2.3 Technical Challenges Identified by DoD	8
2.4 Analysis of DoD Findings: <i>Overlooked Technology Opportunities</i>	11
2.4.1 Existing Intelligent Software in Military Domains	12
2.4.2 AI Application Opportunities for <i>Unmanned Vehicles</i>	13
2.5 The Need to Couple <i>Unmanned Vehicles</i> with AI Technology	14
3. AI Technology Underpinnings	17
3.1 Information-Centric vs. Data-Centric	17
3.2 Service-Oriented Architecture (SOA)	18
3.3 SOA Implementation Components	19
3.3.1 Services Management Framework (SMF)	19
3.3.2 Enterprise Service Bus (ESB)	20
3.3.3 What are Services?	22
3.3.4 Typical Service Requester and Service Provider Scenario	22
3.4 The Need for Information-Centric Software	23
3.5 Layered Architecture	24
4. Intelligent Software Agents	27
4.1 Software Agents as Intelligent Decision-Assistance Tools	28
4.2 Planning and Re-Planning Functions	31
4.3 Truth Maintenance Approach to re-Planning	32
4.4 Learning as Part of a Planning System	32
4.5 Service Agents	32

4.6	Mentor Agents	33
5.	Web Services	35
5.1	Web Services as an Implementation of the SOA Concept	35
5.2	Simple Object Access Protocol (SOAP)	35
5.3	Universal Description, Discovery and Integration (UDDI)	36
5.4	Web Services Description Language (WSDL)	36
5.5	Federated Web Services	36
5.6	Web Services Security	36
6.	Interoperability Bridges	39
6.1	Problem Definition	39
6.2	Solution Criteria	40
6.3	Technologies Employed	41
7.	UAV Image Interpretation: <i>An Example Demonstration</i>	43
7.1	Object Tracking	43
7.2	The Adaboost Algorithm	43
7.3	Online Adaboost	44
7.4	Features for Tracking	45
7.5	Tracking as Classification	46
7.6	Detecting the Object of Interest	47
7.7	Applications to Unmanned Aerial Vehicles	47
8.	Phase II Proposal	49
8.1	Objectives of the UVORIS Proof-of-Concept System	49
8.2	Technical Approach	50
8.2.1	Case-Based Reasoning	50
8.2.2	Ontology-Based Representation	50
8.2.3	Rule-Based Agents	51
8.2.4	Statistical Object Recognition Methodologies	52
8.2.5	Semantic Database Search Techniques	52
8.2.6	Knowledge Management Enterprise Services (KMES [®])	54
8.3	Phase II Task and Deliverables	55
8.4	Estimated Phase II Duration and Cost	57
9.	Acronyms and Definition of Terms	59
10.	References and Bibliography	67

Intelligent Software Agent Technology for Unmanned Aircraft Systems (UAS)

A Feasibility Study

1. Introduction: *Intelligent Software as a Paradigm Shift.*

A very powerful new type of information systems technology is rapidly emerging, driven by government and commercial needs for *expert decision-support* and *knowledge management*. One very apparent result of this technology is increasingly intelligent software systems. Computer programs with collaborative agents that are capable of automatically reasoning about data and the dynamic changes in data that occur in real world decision-making situations are already in use by the military and are now transitioning to the commercial world.

It can be argued that our human view of computer software has been shortsighted in respect to two popular notions: first, that data and information are essentially synonymous terms; and, second, that computer intelligence is largely a misnomer because computers are machines. Neither of these notions is accurate. While we human beings are able to convert data (i.e., numbers and words without relationships) automatically into information due to the experience (i.e., *context*) that is held in our cognitive system, computers do not have the equivalent of a human cognitive system and therefore store data simply as the numbers and words that are entered into the computer. For a computer to interpret data it requires an information structure that provides at least some level of *context*. This can be accomplished utilizing an ontology of objects with characteristics and a rich set of relationships to create a virtual version of real world situations and provide the context within which agent logic can automatically operate.

In the broadest sense an agent is a computer-based program or module of a program that has communication capabilities to external entities and can perform some useful tasks in at least a semi-autonomous fashion. Agent software can range from simple, stand-alone, predetermined applications to the most intelligent, integrated, multi-agent decision-support system that advanced technology can produce today.

There are many types of software agents, ranging from those that emulate symbolic reasoning by processing rules, to highly mathematical pattern matching neural networks, genetic algorithms, and particle swarm optimization techniques. The focus of many of the products of this technology is on ontology-based decision-support systems that utilize agents with symbolic reasoning capabilities. In these systems the reasoning process relies heavily on the rich representation of objects and their relationships provided by the ontology.

The capabilities of ontology-based multi-agent systems are several orders above those of past data-processing systems that were confined to predetermined algorithmic solution sequences. Those systems worked well when the problem in the real world was exactly as predicted during the design and development stages of the software. However, more often than not the problems encountered in the real world did not conform to those predictions. Agent-based programs are able to *adapt* their solution capabilities to a real world problem situation because of the ability of the software agents to reason within the context of the problem situation.

There are essentially two compelling reasons why computer software must increasingly incorporate more and more *intelligent* capabilities. The first reason relates to the current data-processing bottleneck. Advancements in computer technology over the past several decades have made it possible to store vast amounts of data in electronic form. Based on past manual information handling practices emphasis was placed on storage efficiency rather than processing effectiveness. Typically, data file and database management methodologies focused on the storage, retrieval and manipulation of data transactions, rather than the *context* within which the collected data would later become useful in planning, monitoring, assessment, and decision-making tasks.

The second reason relates to the complexity of networked computer and communication systems, and the increased reliance of organizations on the reliability of such information technology environments. Recent studies conducted by IBM Corporation and others have highlighted the need for autonomic computing as the organizational expectations and dependence on information services leads to more and more complex networked computer solutions (Ganek and Corbi 2003). In the commercial sector “...it is now estimated that at least one-third of an organization’s IT (Information Technology) budget is spent on preventing or recovering from crashes” (Patterson et al. 2002). Simply stated, autonomic computing utilizes the *understanding* that can be represented within an information-centric software environment to allow systems to automatically: (1) reconfigure themselves under dynamically changing conditions; (2) discover, diagnose, and react to disruptions; (3) maximize resource utilization to meet end-user needs and system loads; and, (4) anticipate, detect, identify, and protect themselves from external and internal attacks.

Clearly, both the data bottleneck problem and the need for autonomic computing capabilities are equally applicable to the unmanned vehicle domain. A single unmanned airborne vehicle can easily collect several terabytes of data in a single day. In the case of combined operations involving multiple unmanned airborne vehicles in combination with unmanned ground vehicles and unmanned maritime systems the quantity of data that must be managed and interpreted is well beyond the means of legacy data-centric software. In addition, the operational components of the unmanned systems must be monitored and protected from active security threats. It would appear that all of the autonomic capabilities listed above are essential for the reliable operation of unmanned vehicles.

1.1 Purpose and Structure of Report

The purpose of this report is to explore the software needs of unmanned vehicles and investigate whether intelligent software agent technology incorporating artificial intelligence methodologies would be able to satisfy some or all of these needs.

Section 2 briefly defines the different types of unmanned vehicles, describes their principal military application areas, and summarizes the existing functional limitations that have been identified in recent US Department of Defense studies. The authors believe that a strong case is made at the end of this Section for the highly beneficial application of intelligent agent technology to unmanned vehicles. Sections 3, 4, 5, and 6 provide an explanation of information-centric, service-oriented architecture concepts, and more detailed descriptions of the principles of intelligent agents, web services, and interoperability bridges, respectively. Section 7 describes a demonstration of automated object interpretation as an example of just one application of intelligent agents to unmanned vehicles.

2. Background: *Unmanned Aircraft Systems (UAS)*

Unmanned vehicles are being increasingly used for both military and non-military purposes. Military uses focus on: reconnaissance and surveillance; target identification and designation; counter-mine warfare; and, chemical, biological, radiological, nuclear, explosive (CBRNE) reconnaissance. Commercial uses include: gathering of climate and weather data; water resource forecasting; ecosystem monitoring; coastal mapping; customs and border protection; search for natural disaster survivors; study of volcanic activities; real estate photography; and so on.

2.1 Definitions

Today, *unmanned vehicles* are being used by all military services to varying degrees. However, it is generally recognized that their use will increase at an accelerating rate over the next two decades. The Floyd D. Spence National Defense Authorization Act for Fiscal Year 2001¹ stated two overall goals in respect to the development and deployment of *unmanned vehicle* capabilities, namely: (a) by 2010, one third of the aircraft in the operational deep strike force should be unmanned; and, (b) by 2015, one third of the Army's Future Combat Systems (FCS) operational ground combat vehicles should be unmanned. The use of unmanned vehicles by the services is based on mission responsibilities and operational needs, as follows:

Air Force: Mostly *Unmanned Aircraft Systems (UAS)*, also commonly referred to as Unmanned Aerial Vehicles (UAV) although strictly speaking UAVs are a subset of UASs.

Army: Mostly *UAS* and *Unmanned Ground Vehicles (UGV)*.

Navy: Mostly *UAS*, *Unmanned Maritime Systems (UMS)* and *Unmanned Undersea Vehicles (UUV)*.

Marine Corps: Mostly *UAS* and *UGV*.

The US Department of Defense (DoD) defines *unmanned vehicles* according to Joint Publication 1-02 (DoD 2001), as follows:

Unmanned Vehicle: *A powered vehicle that does not carry a human operator, can be operated autonomously or remotely, can be expendable or recoverable, and can carry a lethal or non-lethal payload. Unmanned vehicles are the primary component of Unmanned Systems.*

According to this definition ballistic or semi-ballistic vehicles, cruise missiles, artillery projectiles, torpedoes, mines, satellites, and unattended sensors (with no form of propulsion) are not considered to be unmanned vehicles.

The military vision of DoD is to standardize the terminology, control systems and communication systems used by *unmanned systems*, and issue standards for the design, manufacture, testing, and performance of *unmanned systems*. DoD is planning to spend between \$3 and \$4 billion per year on *unmanned systems* between 2009 and 2013.

¹ Section 220 of the Floyd D. Spence National Defense Authorization Act for FY2001 (Public Law 106-398; 114 Stat 1654A-38).

The public and commercial vision of the Federal Aviation Administration (FAA) is to issue a set of uniform regulations for the deployment of UASs in the air space. The FAA is planning to spend less than \$1 billion on UAS projects over the next five years.

2.2 Primary Military Applications

As of October 2006, coalition UASs had flown almost 400,000 hours, UGVs had responded to over 11,000 Improvised Explosive Device (IED) situations, and UMSs had been employed to provide port security in support of Operations Enduring Freedom and Iraqi Freedom (OSD 2007).

Unmanned vehicles (i.e., UAS, UAV, UGV, UMV, and UUV) are intended to contribute to filling military capability gaps that have been identified in various recent studies². Of the 99 synthesized highest priority capability gaps 17 could be potentially addressed by *unmanned vehicles*. Currently primary military applications of *unmanned vehicles* include:

Reconnaissance: All military services, across the three unit echelons (i.e., company, battalion, and division) and all types of *unmanned vehicles*, listed some form of reconnaissance (i.e., electronic or visual) as the number one priority. The ability to conduct surveillance in hostile areas while maintaining a degree of covertness is a key military requirement. Satellites, manned aircraft and submarines, and unattended sensors all have shortcomings that can be mitigated by *unmanned vehicles*.

Target identification and designation: Identifying, locating and tagging potential targets is a highly suitable task for *unmanned vehicles* that obviates the need for placing warfighters at risk.

Counter mine warfare: The use of UAVs to detect and destroy Improvised Explosive Devices (IEDs) and land mines, as well as the use of UUVs to detect and destroy sea mines is highly desirable.

Chemical, biological, radiological, nuclear, explosive reconnaissance: The ability to find and destroy chemical and biological agents and to survey the extent of contaminated areas is a crucial capability.

2.3 Technical Challenges Identified by DoD

The most critical near-term technical challenge facing *unmanned vehicles* is the development of an autonomous capability to *detect, assess and respond* to near-field objects in their path of travel. For a UAV the near-field can extend to several nautical miles in all directions, while for a UGV or UUV the distance at which an object must be detected is likely to be less than 100 yards.

The technology developments and capability requirements that impact *unmanned vehicles* have been classified into *push factors* (i.e., recent or emerging technology advances and breakthroughs), *pull factors* (i.e., social and cultural issues that determine to what extent and how quickly emerging new technology is accepted by a particular community of interest), and *contextual factors* (i.e., organizational, legal, regulatory, and economic issues that impact the development of new

² The Combatant Commands (COCOMs) submitted 112 capability gaps in their FY2008-13 Integrated Priority List (IPL). Combination with military department identified gaps, Contingency Plan (CONPLAN) 7500, and other lessons learned in the Global War on Terrorism (GWOT) produced 526 capability gaps.

technology)³.

Push factors: The following *push factors* are listed in both a prominent National Research Council study (NRC 2002) prepared at the request of the National Institute for Standards and Technology (NIST) and the definitive Unmanned Systems Roadmap 2007-2032 published by the Office of the Secretary of Defense (OSD 2007) in December 2007:

- Transgenic biopolymers for ultra-lightweight, ultra-strength, flexible, and low-observable membranes (i.e., for airframes and cowlings).
- Nanoparticles such as carbon nanotubes, which could provide mechanical devices with very high resonant frequencies for use in communication links.
- Microelectromechanical systems (MEMS) that could radically reduce the size of *unmanned vehicles* to fly-sized and ant-sized devices.
- Proton exchange membrane fuel cells that already offer power units equivalent to the internal combustion engine, but are characterized by much quieter operation and less complexity.
- Smart materials and structures with the ability to adapt themselves to environmental changes and repair themselves in case of damage.
- Magnetic nanoparticles that may provide the basis of a new generation of data storage devices with a potential storage density of 1,000 gigabits per square inch.

While intelligent software capabilities are mentioned in the OSD report, they are not considered to be advancing at an adequate rate for application to *unmanned vehicles* during the next decade. Instead, the report suggests that advances in computer processor speed and storage density (i.e., memory) will bring improvements in the integration and interpretation of data from sensors.

Pull Factors: Driven by the mission needs of Combatant Commands (COCOMs) the importance of *unmanned vehicles* operating together in a collaborative manner is identified by DoD as an important near-term requirement. DoD envisions UAS, UVG and UMS assets teaming together in *combined arms roles* to augment and extend the role of manned assets. The following *pull factors* are recognized by OSD's Unmanned Systems Roadmap 2007-2032 report (DoD 2007):

1. **Autonomy** and **control** of *unmanned vehicles* in respect to the perception of topographical features and obstacles by UASs flying at low altitude, and cooperative coordination among multiple *unmanned vehicles* of the same and different types.
2. **Bandwidth** issues arising from the use by manufacturers of lower cost COTS data link equipment for *unmanned vehicles* that are designed to use

³ Report prepared by the National Research Council for the National Institute of Standards and Technology: 'Future R&D Environments: A Report for the National Institute of Standards and Technology'; National Academic Press, National Academy of Sciences, Washington, DC, 2002.

the US authorized lowest priority frequency band. Also, this frequency band may be prohibited in some countries.

3. The ***data presentation*** to the human users of the data is identified by OSD as an important consideration. The data must be presented in a manner that extends human perception, reasoning, decision-making, and action capabilities.
4. Standardization of a ***common control approach*** for *unmanned vehicles* is desirable to minimize the proliferation of different hardware and software, which places a severe burden on operator training requirements and increases the risk of human error.
5. Improvements in the ***reliability*** and ***required bandwidth*** of the communication link between the *unmanned system* and the receiver, particularly when multiple networked *unmanned vehicles* are involved.
6. Interconnectivity and interoperability between ***networked unmanned vehicles*** that are operating as a distributed and scalable combat force. Such a combat force could include all types of *unmanned vehicles*, as well as sensors, weapons, command and control (C2) systems, and manned vehicles, and will require data fusion and decision-making assistance capabilities.
7. Data interfaces that allow *unmanned vehicles* to operate both in a closed loop with the control station and as ***network nodes*** that are capable of providing communication services when available and needed.
8. Dynamic obstacle, interference, and ***collision avoidance*** capabilities are required to allow *unmanned vehicles* to operate as autonomous entities in relative safety.
9. Despite the implications of their name, *unmanned vehicles* must be able to effectively integrate with human systems. This requires attention to all aspects of ***Human Systems Integration (HIS)*** including: user-interface design; selection and training of human operators; identification and implementation of process changes; and, graceful degradation of the operational environment.
10. ***Launch and recovery*** operations must be able to be easily performed with safety and a minimum number of personnel.
11. ***Power systems*** are a major factor in the design and efficiency of *unmanned vehicles*. There is a need to minimize the cost, size, and signature of both the energy system and the propulsion unit of *unmanned vehicles*. The desire for long endurance, payload power, and high speed requires increased energy capacity.
12. ***Computer technology*** advances are required if entire UAS missions are to be conducted with little or no human intervention. OSD is looking mostly to hardware advances leading to new computer technologies (e.g., quantum computers) to provide the necessary processor speed and

memory capacity. Again, it is surprising that the report does not recognize the ability of existing Artificial Intelligence (AI) applications to provide at least a starting point for the realization of this objective.

13. **Engineering design** of *unmanned vehicles* should not be solely dictated by technology developments but place equal weight on overall system considerations such as cost, interoperability, size, and number of vehicles to be used.
14. **Reliability** of *unmanned vehicles* is an important factor that influences affordability, availability, and acceptance by warfighters. It is an all-embracing factor that extends beyond the mechanical reliability of the airframe and propulsion unit to the on-board software, the sensors and the dependability of the communication link.
15. **Sensor** research is required to develop new tracking techniques, increase the Area Coverage Rate (ACR), and in general improve sensor classification and identification capabilities. In particular, there is a need for more effective CBRNE and MCM sensors.
16. **Survivability** of *unmanned vehicles* under combat conditions should be a key consideration during the design stage. Similar to the *reliability factor* (see item 14) this is an all-encompassing factor that refers to the total system (e.g., includes the ability of the airframe to withstand air turbulence, the sensors to continue to operate under less than favorable conditions, and the communication link to recover after temporary disruption).
17. **Weaponization** of *unmanned vehicles* is still a highly controversial issue due to reliability and communication link concerns. While rules of engagement may not allow *unmanned vehicles* to utilize on-board weapons without a *man-in-the-loop*, there remains the challenge of the ability of the remote human operator to verify the absolute safety of the unmanned weapon or unarm, destroy or scuttle the weapon.

2.4 Analysis of DoD Findings: *Overlooked Technology Opportunities*

Although an autonomous capability has been identified by virtually all DoD studies as the most critical technical need for *unmanned vehicles*, software incorporating AI methodologies has not been recognized by these studies as a key *push factor* that will become available during the next decade. The Unmanned Systems Roadmap 2007-2032 study specifically precludes advances in speech recognition and visual recognition from being enabling technologies for *unmanned vehicles* for at least another decade (OSD 2007, 47). Also in these studies the need for autonomous *detect, assess and respond* capabilities for *unmanned vehicles* to avoid obstacles and threats is recognized. However, again, the solution path suggested by DoD is based on improvements in sensor technology rather than intelligent software capabilities. Even the necessary and potentially powerful combination of physical sensors with intelligent software systems does not yet appear to be adequately recognized by DoD in its various *unmanned vehicles* studies.

The issues is not that DoD is altogether overlooking the role that AI-based software could play to

overcome the limitations that currently impede the ability of *unmanned vehicles* to realize their full potential, but that AI methodologies will require another decade or more to become a practical proposition. In fact, the promise that AI technologies hold for *unmanned vehicles* is clearly stated in several sections of OSD's Unmanned Systems Roadmap 2007-2032 study. However, the ability to apply this technology in the near term is explicitly dismissed. This is a surprising conclusion in light of the level of maturity that many AI-based methodologies have reached and the degree to which intelligent software has over the past eight years penetrated both military and commercial information management.

2.4.1 Existing Intelligent Software in Military Domains

Ontology-based software applications with rule-based agents have been successfully applied in both tactical command and control (C2) and logistical planning and execution since the late 1990s. Examples include the Integrated Marine Multi-Agent Command and Control System (IMMACCS) developed under sponsorship of the Marine Corps Warfighting Laboratory, which was successfully tested in 1999 as the C2 system of record in the Urban Warrior Advanced Warfighting Exercise (Pohl et al. 1999). Subsequently, in 2002, IMMACCS was evaluated by two independent teams during a Limited Technical Assessment (LTA) to determine its ability to maintain a common tactical picture in a wireless communication environment operating within extremely limited bandwidth conditions (SPAWAR 2002). It was found that an information-centric⁴ software system, such as IMMACCS, requires less bandwidth than a data-centric system in which there is no internal understanding of the meaning of the data being exchanged and processed.

Other systems with intelligent agents have followed: SEAWAY for logistical sea-basing operations (Wood et al. 2000); SILS for shipboard readiness assessment (Zang and Neff 2003); TRANSWAY for goods movement and distribution planning (Nibecker et al. 2006); ICODES for ship load-planning (Goodman and Pohl 2003, Diaz et al. 2006); and, COSMOS for the secure exchange of data among coalition forces based on the Multilateral Interoperability Programme (MIP) data exchange model⁵ (Leighton and Undesser 2008).

For example, the Integrated Computerized Deployment System (ICODES) has been the system of record for ship load-planning for the past 10 years. Released by USTRANSCOM in 1997 as a *migration system*, ICODES is used today at more than 50 ports world-wide with a military user-base of close to 3,000. In 2007, ICODES Global Services (GS) was designated to become the Single Load-Planning Capability (SLPC) for all conveyance types (i.e., ships, airlift, rail, truck

⁴ The term *information-centric* refers to the ability of software to have some level of understanding of the data that are being processed by the software. This understanding is derived from a virtual internal model of the real world context in which the data exist. The approach for constructing this virtual model within the constraints of current information technology is an ontology, which is rich in relationships between entities that are represented as objects with behavioral characteristics.

⁵ The principal objective of the Coalition Secure Management and Operations System (COSMOS) Advanced Concept Technical Demonstration (ACTD) is to provide US and coalition forces with a secure and reliable capability for sharing the right information at the right time with the right forces. The solution that COSMOS provides is built on the internationally recognized C2IEDM (JC3IEDM) data structure. It extends this data model with an ontology that provides the necessary context for software agents to automatically reason about battlespace events and assist the Information Management Officer (IMO) with the expeditious policy-driven exchange of information in a cross-domain environment.

convoys, and marshalling yards) by 2010⁶. The ICODES agents monitor the principal determinants of cargo stowage, including: the placement and segregation requirements for hazardous cargo items; the trim and stability requirements of the conveyance; the accessibility of stow areas; the correct placement of cargo items in respect to restricted areas and inter-cargo spacing tolerances; and, the accuracy of cargo characteristics (e.g., dimensions, weight, type, and identification codes) relative to standard cargo libraries and associated reference tables. Through its embedded AI capabilities ICODES has been able to radically reduce the time required for cargo load-planning operations and greatly increase the quality of load-plans, in terms of accuracy, anticipation of conflicts, and an automatic near real-time re-planning capability.

2.4.2 AI Application Opportunities for *Unmanned Vehicles*

At least six of the technology *pull factors* listed in OSD's Unmanned Systems Roadmap 2007-2032 report (OSD 2007) are very much dependent on software with AI capabilities. Certainly the need for *autonomy* and *control* (see items 1 and 8 in Section 2.3 above) is greatly dependent on intelligent software systems that are capable of the near real-time detection, location, and identification of physical objects, as well as the rapid formulation and implementation of appropriate actions. It should be noted that throughout the UAS literature the need for autonomous *detect, assess and respond* capabilities is cited as one of the highest priority requirements for *unmanned vehicles*. In the commercial and public world, this is the critical factor that will determine whether UASs will gain widespread FAA certification to operate in the national airspace (GAO 2008).

The need for *reliability* (item 14) and *survivability* (item 16), likewise will require intelligent software that apart from providing inferencing and decision-making capabilities is also capable of continuously monitoring both the internal *unmanned vehicle* components and the external environmental conditions. Integral to meeting these requirements is the ability to interface with the human controller in an appropriate *data presentation* (item 3) format. The automated determination of what data should be presented to the remote controller, at what level of urgency and when, will require the software to have some understanding of the context of the situation. AI methodologies for representing context in software, such as ontology domains, have reached an acceptable level of maturity in recent years.

The desire to operate multiple *unmanned vehicles* of various types (i.e., UASs, UVGs, UMSs, and UUVs) in a networked, collaborative, combined arms environment increases the degree of complexity significantly. The attendant *interconnectivity* and *interoperability* requirements (item 6) call for automated data interpretation and information management capabilities that can be met only by the application of AI methodologies. In particular, this will require the representation of context in an ontology format so that software agents are able to correctly interpret the continuous flow of operational data and generate plans for appropriate courses of action. The fact that these plans will need to be synchronized within a community of *unmanned vehicles* increases the potential complexity by an order of magnitude.

In all of these cases the capabilities required by *unmanned vehicles* require the vehicle, whether in the air (UAS), on the ground (UGV), in water (UMS), or under water (UUV), must be at least partially provided by non-human means. Even where *unmanned vehicles* are subject to remote

⁶ In November USTRANSCOM's Distribution Steering Group (DSG) designated ICODES to become the Single Load Planning Capability (SLPC) for all types of conveyances, with a scheduled release date of 2010.

human control, the human operator cannot have complete knowledge or situation awareness of the precise circumstances that apply to the vehicle. For example, in the case of a UAV, the local wind conditions in immediate proximity of the aircraft may not be able to be judged correctly by a remote human controller who has to depend on the interpretation of regional weather data. Even more critical can be the need for the UAV to promptly recognize and, if necessary, quickly react to other flying objects in its vicinity. The ability to differentiate in near real-time between a bird and another UAV may be a critical determination that cannot be delayed by a remotely performed analysis based on incomplete information.

2.5 The Need to Couple *Unmanned Vehicles* with AI Technology

It is not being suggested that AI technology has reached the stage where computer software can perform as well as an alert and skilled human decision maker. In fact, it is most unlikely that computers will ever be able to completely emulate human intelligence. However, there is a need for AI capabilities and software tools to be developed in close alignment with the real world devices and circumstances in which they are expected to be employed. AI technology is certainly sufficiently advanced and mature for it to be considered as an important enabling capability for all types of *unmanned vehicles*. Rather than wait for it to mature further, it is highly desirable for the developers of *unmanned vehicles* and intelligent software systems to work together in close collaboration. Experience has shown that the exploitation of new capabilities occurs incrementally over time as the users of the capabilities fine tune the capabilities and the capabilities reshape the processes that they support. Therefore, it is important for the development of more powerful *unmanned vehicle* capabilities to be closely coupled with advances in AI technology. Every effort should be made to encourage these two research communities to work together, so that the current weaknesses of *unmanned vehicles* can be addressed with the most appropriate and effective intelligent software solutions.

Not all of the weaknesses of *unmanned vehicles* are related to the limited autonomous capabilities of the vehicles themselves. The timely processing and appropriate sharing of the data collected by each vehicle has been identified by the users of *unmanned vehicles* as a serious bottleneck (Erwin 2008). For example, a single UAV may collect video streams through 100 mega-pixel cameras operating at several frames per second. The total volume of data collected by this one UAV, which must be stored and processed, may exceed 100 terabytes per day. In combined arms operations several UAVs may be networked with surface *unmanned vehicles* such as UGVs and UMSs. This potentially increases the volume of data by at least one order of magnitude. To be useful, the data collected must be rapidly analyzed and interpreted so that timely decisions can be made. However, the available legacy data-processing tools are inadequate for handling such large quantities of data. The situation is exacerbated in joint operations, where there are serious horizontal interoperability problems standing in the way of data sharing.

AI tools are available for the automated analysis and interpretation of data. Under the general category of natural language processing these tools and methodologies include: case-based classification for the automated processing of terminology differences and ambiguities in textual data; information gain techniques for selecting and weighting features based on the information theory principle that attaches more importance to rarely occurring items than frequently occurring items; semantic equivalence determination through the automated construction of synonym libraries; abbreviation resolution based on the detection of character order patterns; model-based reasoning for coordinating the activities of multiple agents and amalgamating their inferences into

composite semantic equivalence scores; and, several other AI-based methodologies and techniques.

The lack of interoperability among existing software systems is a problem that is not unique to the processing of data in *unmanned vehicle* systems. It has been identified as a major obstacle to effective information management by all communities of interest. The application of Service-Oriented Architecture (SOA) concepts and principles allow the implementation of enterprise-wide information management environments in which *interoperability* is a primary design criterion rather than an afterthought.

3. AI Technology Underpinnings

An agile planning capability requires both the ability to interpret data in context and the flexibility to provide access to decision-support tools regardless of whether these are part of the same application or another application. The definition of agility as the ability to rapidly adapt to changing conditions has two implications. First, in a real world environment the operational data that enter a particular application may not adhere exactly to the specifications on which the design of the software was originally based. An agile software application will therefore need to have the ability to automatically interpret the incoming data within the appropriate context and make the necessary processing adjustments. Second, under such dynamic conditions it is likely that the user will have a need for tools that were not foreseen during the design of the application and are therefore not available. An agile software environment will therefore have to provide access to a wide range of tools, at least some of which may not be an integral component of the particular application that the operator is currently using. This suggests a system environment in which software tools can be seamlessly accessed across normal application domain boundaries.

3.1 Information-Centric vs. Data-Centric

There are several reasons why computer software must increasingly incorporate more and more *intelligent* capabilities (Pohl 2005). Perhaps the most compelling of these reasons relates to the current data-processing bottleneck. Advancements in computer technology over the past several decades have made it possible to store vast amounts of data in electronic form. Based on past manual information handling practices and implicit acceptance of the principle that the interpretation of data into information and knowledge is the responsibility of the human operators of the computer-based data storage devices, emphasis was placed on storage efficiency rather than processing effectiveness. Typically, data file and database management methodologies focused on the storage, retrieval and manipulation of data transactions, rather than the *context* within which the collected data would later become useful in planning, monitoring, assessment, and decision-making tasks.

The term *information-centric* refers to the representation of information, as it is available to software modules, 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, folded, bound into volumes, and so on. As such, representation can exist at varying levels of abstraction. The lowest level of representation considered 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-centric 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 an

empowering design principle that allows software to undertake the interpretation of operational data changes within the context provided by the internal information model (i.e., ontology).

Even before the advent of the Internet and the widespread promulgation of SOA concepts it was considered good software design and engineering practice to build distributed software systems of loosely coupled modules that are able to collaborate by subscription to a shared information model. The principles and corresponding capabilities that enable these software modules to function as decoupled services include:

- An internal *information* model that provides a usable representation of the application domain in which the service is being offered. In other words, the context provided by the internal information model must be adequate for the software application (i.e., service) to perform as a useful adaptive set of tools in its area of expertise.
- The ability to *reason* about events within the context provided by the internal information model. These reasoning capabilities may extend beyond the ability to render application domain related services to the performance of self-monitoring maintenance and related operational efficiency tasks.
- Facilities that allow the service to *subscribe* to other internal services and understand the nature and capabilities of these resources based on its internal information model.⁷
- The ability of a service to understand the notion of *intent* (i.e., goals and objectives) and undertake self-activated tasks to satisfy its intent. Within the current state-of-the-art this capability is largely limited by the degree of context that is provided by the internal information model.

Additional capabilities that are not yet able to be realized in production systems due to technical limitations, but have been demonstrated in the laboratory environment, include: the ability of a service to *learn* through the acquisition and merging of information fragments obtained from external sources with its own internal information model (i.e., dynamically *extensible* information models); extension of the internal information model to include the internal operational domain of the software application itself and the role of the service within the external environment; and, the ability of a service to increase its capabilities by either *generating* new tools (e.g., creating new agents or cloning existing agents) or automatically *searching* for external assistance.

3.2 Service-Oriented Architecture (SOA)

The notion of *service-oriented* is ubiquitous. Everywhere we see countless examples of tasks being performed by a combination of services, which are able to interoperate in a manner that results in the achievement of a desired objective. Typically, each of these services is not only *reusable* but also sufficiently *decoupled* from the final objective to be useful for the performance of several somewhat similar tasks that may lead to quite different results. For example, a common knife can be

⁷ This must be considered a minimum system capability. The full implementation of a web services environment should include facilities that allow a service to *discover* other external services and understand the nature and capabilities of these external services.

used in the kitchen for preparing vegetables, or for peeling an orange, or for physical combat, or as a makeshift screwdriver. In each case the service provided by the knife is only one of the services that are required to complete the task. Clearly, the ability to design and implement a complex process through the application of many specialized services in a particular sequence has been responsible for most of mankind's achievements in the physical world. The key to the success of this approach is the *interface*, which allows each service to be utilized in a manner that ensures that the end-product of one service becomes the starting point of another service.

In the software domain these same concepts have gradually led to the adoption of Service-Oriented Architecture (SOA) principles. While SOA is by no means a new concept in the software industry it was not until Web services came along that these concepts could be readily implemented (Erl 2005). In the broadest sense SOA is a software framework for computational resources to provide services to customers, such as other services or users. The Organization for the Advancement of Structured Information (OASIS)⁸ defines SOA as a “... *paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains*” and “...*provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects with measurable preconditions and expectations*”. This definition underscores the fundamental intent that is embodied in the SOA paradigm, namely *flexibility*. To be as flexible as possible a SOA environment is highly modular, platform independent, compliant with standards, and incorporates mechanisms for identifying, categorizing, provisioning, delivering, and monitoring services.

In a networked *unmanned vehicle* information management environment, SOA-based software system design principles will allow large volumes of data of different formats and adhering to different schemas to be exchanged between services, mapped to the appropriate context, and automatically interpreted by intelligent agents. While the available technology for constructing such software environments is not flawless, it is nevertheless sufficiently mature to encourage immediate implementation.

3.3 SOA Implementation Components

The principal components of a conceptual SOA implementation scheme (Figure 3.1) include a Services Management Framework (SMF), various kinds of foundational services that allow the SMF to perform its management functions, one or more portals to external clients, and the enterprise services that facilitate the ability of the user community to perform its operational tasks.

3.3.1 Services Management Framework (SMF)

A Services Management Framework (SMF) is essentially a SOA-based software infrastructure that utilizes tools to manage the exchange of messages among enterprise services. The messages may contain requests for services, data, the results of services performed, or any combination of these. The tools are often referred to as foundational services because they are vital to the ability of the SMF to perform its management functions, even though they are largely hidden from the user community. The SMF must be capable of:

⁸ OASIS is an international organization that produces standards. It was formed in 1993 under the name of SGML Open and changed its name to OASIS in 1998 in response to the changing focus from SGML (Standard Generalized Markup Language) to XML (Extensible Markup Language) related standards.

- Undertaking any transformation, orchestration, coordination, and security actions necessary for the effective exchange of the message.

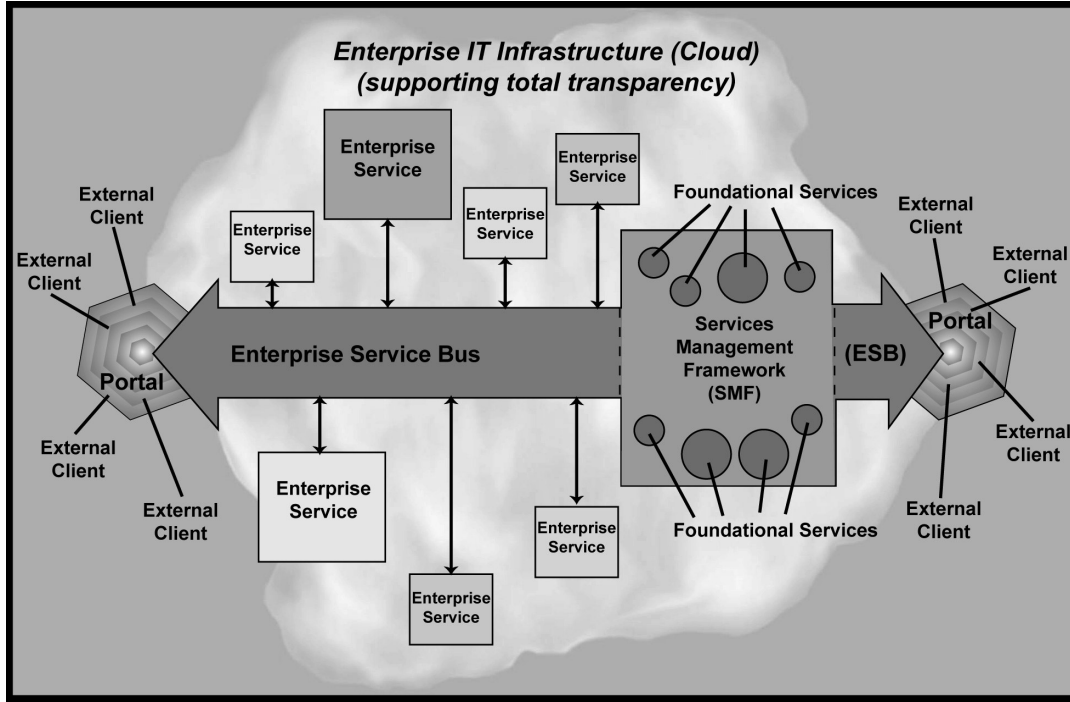


Figure 3.1: Principal components of a conceptual SOA implementation

- Maintaining a loosely coupled environment in which neither the service requesters nor the service providers need to communicate directly with each other; - or even have knowledge of each other.

A SMF may accomplish some of its functions through an Enterprise Service Bus (ESB), or it may be implemented entirely as an ESB.

3.3.2 Enterprise Service Bus (ESB)

The concept of an Enterprise Service Bus (ESB) greatly facilitates a SOA implementation by providing specifications for the coherent management of services. The ESB provides the communication bridge that manages the exchange of messages among services, although the services do not necessarily know anything about each other. According to Erl (2005) ESB specifications typically define the following kinds of message management capabilities:

- **Routing:** The ability to channel a service request to a particular service provider based on some routing criteria (e.g., static or deterministic, content-based, policy-based, rule-based).
- **Protocol Transformation:** The ability to seamlessly transform the sender's message protocol to the receiver's message protocol.
- **Message Transformation:** The ability to convert the structure and format of a message to match the requirements of the receiver.

- **Message Enhancement:** The ability to modify or add to a sender’s message to match the content expectations of the receiver.
- **Service Mapping:** The ability to translate a logical business service request into the corresponding physical implementation by providing the location and binding information of the service provider.
- **Message Processing:** The ability to accept a service request and ensure delivery of either the message of a service provider or an error message back to the sender. Requires a queuing capability to prevent the loss of messages.
- **Process Choreography and Orchestration:** The ability to manage multiple services to coordinate a single business service request (i.e., choreograph), including the implementation (i.e., orchestrate). An ESB may utilize a Business Process Execution Language (BPEL) to facilitate the choreographing.
- **Transaction Management:** The ability to manage a service request that involves multiple service providers, so that each service provider can process its portion of the request without regard to the other parts of the request.
- **Access Control and Security:** The ability to provide some level of access control to protect enterprise services from unauthorized messages.

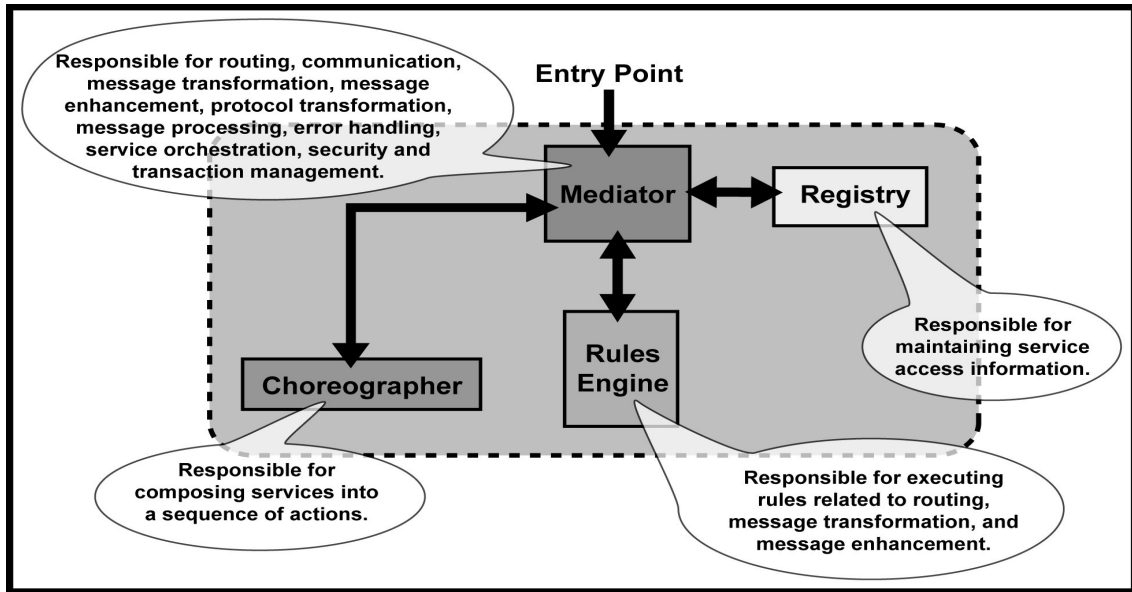


Figure 3.2: Primary ESB components

There are quite a number of commercial off-the-shelf (COTS) ESB implementations that satisfy these specifications to varying degrees. A full ESB implementation would include four distinct components (Figure 3.2): Mediator; Service Registry; Choreographer; and, Rules Engine. The Mediator serves as the entry point for all messages and has by far the largest number of message management responsibilities. It is responsible for routing, communication, message transformation, message enhancement, protocol transformation, message processing, error handling, service orchestration, transaction management, and access control (security).

The Service Registry provides the service mapping information (i.e., the location and binding of each service) to the Mediator. The Choreographer is responsible for the coordination of complex business processes that require the participation of multiple service providers. In some ESB implementations the Choreographer may also serve as an entry point to the ESB. In that case it assumes the additional responsibilities of message processing, transaction management, and access control (security). The Rules Engine provides the logic that is required for the routing, transformation and enhancement of messages. Clearly, the presence of such an engine in combination with an inferencing capability provides a great deal of scope for adding higher levels of intelligence to an ESB implementation.

3.3.3 What are Services?

In a SOA-based system environment *services* are self-contained software components that offer their capabilities to external service requesters. In more technical terms they are defined as "... *a coarse-grained, discoverable software entity that exists as a single instance and interacts with applications and other services through a loosely coupled (often asynchronous), message-based communication model*" (Brown et al. 2003). They are designed to be platform independent and adaptable to a variety of applications. It is this adaptability that promotes their high degree of *reusability*. Some *services* may have quite narrow capabilities such as the reformatting of weather data into a software interpretable weather forecast, while others will incorporate larger functional domains such as the pairing of targets with weapons or the optimum routing of UASs from multiple origins along alternative routes to multiple destinations within the theater. However, all of the *services* that operate within a given application domain are closely aligned to the knowledge context of that domain by sharing the same information model. While the software code of an information-centric *service* is reusable, its internal information model needs to be reconfigured as it is moved from one application domain to another. This ensures that the *services* within any particular application environment are able to exchange data within the functional context of that environment.

3.3.4 Typical Service Requester and Service Provider Scenario

The following sequence of conceptual steps that must be taken by the SMF to support a SOA system environment is not inclusive of every variance that might occur. It is intended to provide a brief description of the principal interactions involved (Figure 3.3).

While the Service Requester knows that the Mediator is the entry point of the ESB component of the SMF and what bindings (protocol) are supported by the Mediator, it does not know which Service Provider will satisfy the request because it knows nothing about any of the other enterprise services that are accessible through the Mediator. Therefore, the conceptual SOA-based infrastructure shown in Figure 3.1 is often referred to as the *Cloud*.

The Mediator is clearly in control and calls upon the other primary components of the ESB if and when it requires their services. It requests the *handle* (i.e., location and mappings) of the potential Service Providers from the Service Registry. If there are multiple Service Provider candidates then it will have to select one of these in Step (6) to provide the requested service. The Mediator will invoke any of the foundational services in the SMF to validate (i.e., access control), translate, transform, enhance, and route the message to the selected Service Provider. The latter is able to accept the message because it is now in a data exchange format that the Service Provider supports.

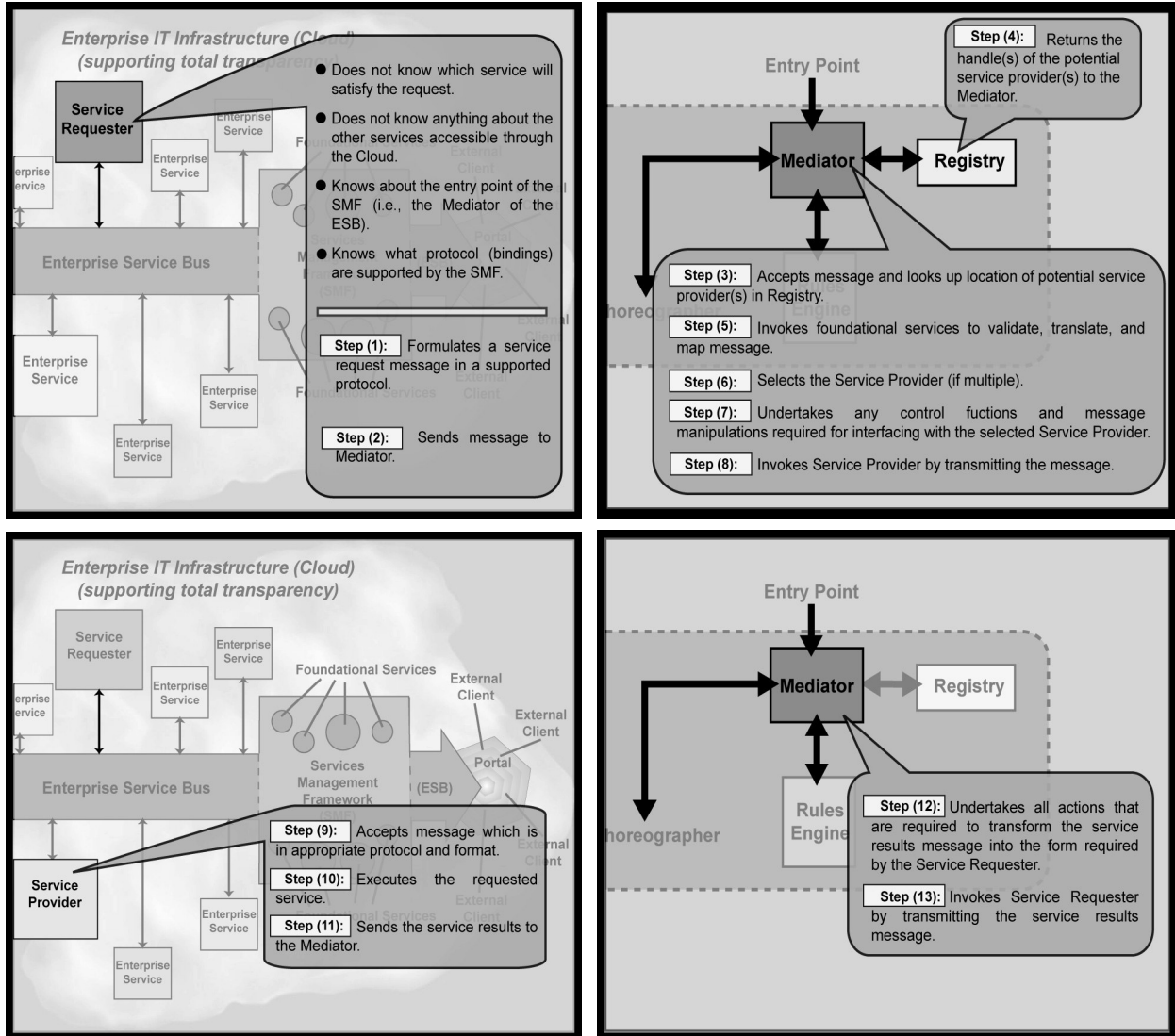


Figure 3.3: Conceptual *Cloud* operations

Similar transformation and mapping actions are taken by the Mediator after it receives the reply message from the Service Provider, so that it complies with the data exchange format supported by the Service Requester. On receiving the response message the Service Requester does not know which service responded to the request, nor did it have to deal with any of the data exchange requirements of the Service Provider.

3.4 The Need for Information-Centric Software

The principal elements or building blocks of an information-centric software environment are data, information, knowledge, and wisdom (Figures 3.4 and 3.5). Data essentially are numbers and words without relationships. We human beings are able to interpret data into information by utilizing the context that we have accumulated in our cognitive system over time (i.e., our experience). Computers do not have a human-like cognitive system and therefore any data stored in a computer will need to be interpreted by the human user. While the computer is able to order, recast,

categorize, catalog, and process the data in many different ways, it cannot use them as the basis of any reasoning sequence. However, if we store not only the data but also a rich set of relationships that place the data into context then it is not difficult to develop software modules (i.e., agents) with reasoning capabilities. In this way it is possible to develop software with some internal understanding of what it is processing (Pohl 2001).

The ability to represent information in computer software has been available for at least the past 30 years (Winston 1970, Biermann and Feldman 1972, Cohen and Sammut 1978). Hampered initially by a lack of hardware power and later by the absence of any compelling need to involve the computer in the direct interpretation of data, these information modeling techniques were not applied in the mainstream of computer software development until fairly recently. The compelling reasons that have suddenly brought them to the foreground are the increasing volume of computer-based data that is beginning to overwhelm human users.

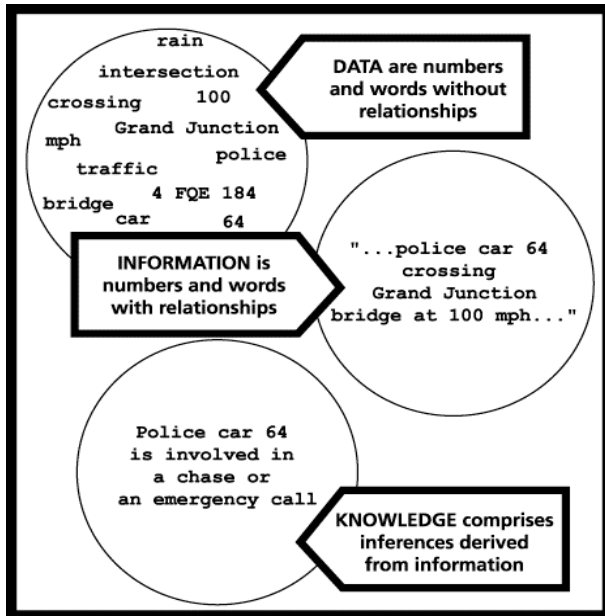


Figure 3.4: Data, information and knowledge

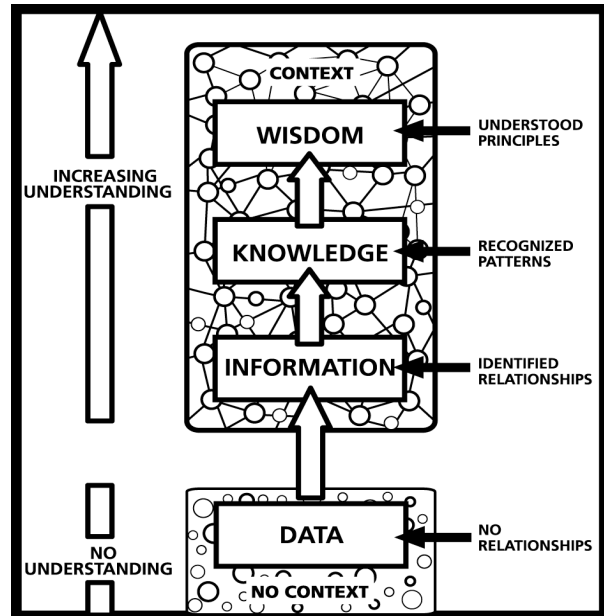


Figure 3.5: Importance of context

The physical gap that is shown schematically between the realms of the data environment without context and no understanding, and the information environment with context and ascending levels of greater understanding in Figure 3.5, is intended to underscore the fundamental difference between the two realms. The transition from data-processing software to information-centric software requires a paradigm shift in the human perception of the role of computers. By incorporating an internal information model (i.e., ontology) that represents portions of real world context as a virtual environment of objects their characteristics and the associations that relate these objects, information-centric software is capable of performing a useful level of automatic reasoning. A number of software agents with relatively simple reasoning capabilities are able to collaborate and through their collective efforts come to more sophisticated conclusions.

3.5 Layered Architecture

Since the early 1970s the ability of computers to store large amounts of data has been increasingly

moderately organized operational data flow. By providing access to both the operational data (Data Portals) and the archived summary data (Data Warehouses) this structured data level represents the integrating data layer that constitutes the bottom layer of a knowledge management system, serving as a necessary foundation for an upper information layer (Figure 3.6). The upper layer utilizes an internal information model (i.e., ontology, Figure 3.7) to provide context for the automatic reasoning capabilities of software agents. Essentially, these agents enabled by their reasoning capabilities constitute a set of intelligent tools that continuously monitor the events (i.e., changes) occurring in the operational environment.

The interface between the lower data-processing layer and the higher information management layer consists of a translation facility that is capable of mapping the data schema of the lower layer to the information representation (i.e., ontology) of the upper layer (Figure 3.6). In this manner, the ontology of the information management layer can be populated with near real-time operational data and archived summary data from Data Warehouses. This mapping process should be bidirectional so that the results of agent actions can be readily transmitted to any data-centric applications that reside in the data layer.

4. Intelligent Software Agents

There are many types of software agents, ranging from those that emulate symbolic reasoning by processing rules, to highly mathematical pattern matching neural networks (McClelland and Rumelhart 1988), genetic algorithms (Koza 1992), and particle swarm optimization techniques (Kennedy and Eberhart 2001). In general terms software agents are defined by Wooldridge and Jennings (1995) as “... *computer systems, situated in some environment, that are capable of flexible autonomous actions ...*”. The three critical words in this definition are situated, flexible, and autonomous. Situated means that the agent receives information from its environment and is capable of performing acts that change this environment. Autonomous refers to the agent’s ability to act without the direct intervention of human users. In other words that the agent has some degree of control over its own actions and internal state. And, flexible means that the system is: responsive - by perceiving its environment and being able to respond in a timely fashion to changes that occur in it; proactive - by exhibiting opportunistic, goal-directed behavior and exercising initiative where appropriate; and, social - by interacting, when appropriate, with other agents and human users in order to complete its own problem solving tasks and help others with their activities.

How do these characteristics of software agents translate to the kind of knowledge management system described above? The agent tools are situated since they receive a continuous flow of operational information generated by the activities of the organization, and perform acts that may change that environment (e.g., creating alerts, making suggestions, and formulating recommendations). The agent tools are autonomous because they act without the direct intervention of human users, even though they allow the latter to interact with them at any time. In respect to flexibility, the agent tools possess the three qualities that define flexibility within the context of the above definition. They are responsive, since they perceive their environment through an internal information model (i.e., ontology) that describes many of the relationships and associations that exist in the real world environment. They are proactive because they can take the initiative in making suggestions or recommendations (e.g., sensor selection and activation by a UAV, *unmanned vehicle* configurations in combined arms missions, or route selection by a UGV based on topographical features) and they do that in an opportunistic fashion. For example, when a request for UAV support is initiated, a Route agent may immediately and without any explicit request from the user, determine the optimum route under current traffic conditions that should be used by the UAV to reach the desired target area.

The ability of software agents to communicate (i.e., socialize) with each other and with human users to work on their own problems or assist others with their problems, is a powerful capability of the information layer in a knowledge management system. It allows several agents to collaborate and concurrently explore different aspects of a problem from multiple points of view, or develop alternative solutions for future negotiation.

Symbolic reasoning agents that are quite common in information management systems incorporate collections of rules that monitor specific conditions and generate alerts when these conditions are satisfied. The general design of such an agent consists of three components: the conditions that trigger the agent (i.e., the functional specification of the agent); the objects and their attributes that are involved in these conditions (i.e., the part of the internal information model (i.e., ontology) that is used by the agent); and, the logic that defines the relationships among these objects and attributes.

One important aspect of autonomy in agent applications is the ability of agents to perform tasks whenever these may be appropriate. This requires agents to be continuously looking for an opportunity to execute. In this context opportunity is typically defined by the existence of sufficient information. For example, to identify a shortage of fuel in a UAV either some agent has to monitor the consumption of fuel until there is a shortage and then issue a warning, or one or more agents collaboratively project that based on developing conditions there is likely to be a shortage of fuel at some specific time in the future.

The requirements for rule-based agents are defined in terms of two elements: conditions; and, actions. The conditions are the specifications of the situation that the agent monitors, while the actions are the alerts that should be generated when these conditions are true. Typically, conditions are specified in terms of objects, attributes and the relationships among them. Each condition is formed by a pattern of object, attributes, values, and Boolean tests. Patterns are grouped by logical connectors, such as AND, OR, and NOT. The more patterns and relationships that are specified, the more specific these conditions become. The right hand side of a rule represents the actions to be taken when the conditions are satisfied. The most general type of action is to generate an alert. However, there are many other kinds of actions that rule-based agents can perform.

4.1 Software Agents as Intelligent Decision-Assistance Tools

There are high expectations that intelligent software agents will solve many of our current information system woes, such as lack of interoperability, multiple failure points, vulnerability to intrusion, making the right information available to the right person at the right time, and proposing solutions under time-critical conditions. Software agents do not have magical human-like capabilities. It is not possible to simply develop a piece of software code that is capable of reasoning about conditions and circumstances like we human beings appear to be able to do. Computers are not human beings and definitely do not have human capabilities. Yet, it is indeed possible to develop software agents that are capable of accomplishing human-like tasks such as recognizing certain conditions, reasoning about these conditions, forming conclusions, and taking actions on the basis of those conclusions.

At first sight the above statements may appear to be contradictory. Software agents do not have human-like capabilities and yet, they are able to accomplish human-like tasks. There is obviously a missing link, a particular ingredient in a software environment that makes it possible for software agents to perform tasks that would normally require human intelligence. Although there is an increasing acceptance of the notion of intelligent computer-based agents, what is not generally understood are the kinds of fundamental capabilities that allow such agents to perform intelligent tasks and the nature of the software environment that is required to support these capabilities?

First we should ask ourselves: What precisely are the capabilities that a software agent needs to have to be able, for example, to determine the information required by a given computer user at any point in time and to prepare alternative solutions for a particular problem situation? Clearly, such tasks require *reasoning* capabilities. The obvious next question then becomes: How can we make it possible for a piece of software code to reason about anything?

To answer this second question we need to examine in some detail what is meant by reasoning. In general terms, reasoning is a logical process involving the systematic interpretation of information. From our earliest school years we learn to assemble information into a form that facilitates various

problem-solving activities. For example, we learn how to extract the few contextually important pieces of information from a passage of text, or how to rearrange a set of conditions to establish a suitable framework for drawing conclusions. In simplest terms this logical process can often be reduced to a set of conditions, the application of certain tests to these conditions, and the drawing of conclusions based on the outcome of the tests. For example (Figure 4.1): I usually commute to work by bicycle. Tomorrow morning I have to be in the office very early for a meeting at 7 am. IF it rains tomorrow THEN I will not commute by bicycle, but use my car instead. IF I have to use my car THEN I will need to leave 20 minutes earlier than normal to be able to find a parking space, and so on.

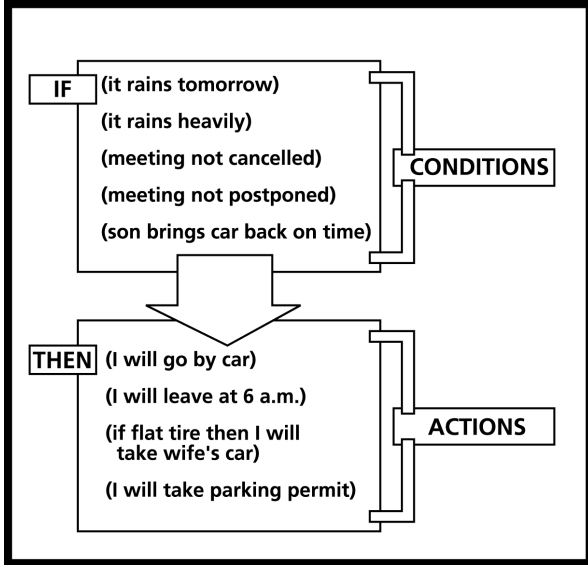


Figure 4.1: Typical rule (or production)

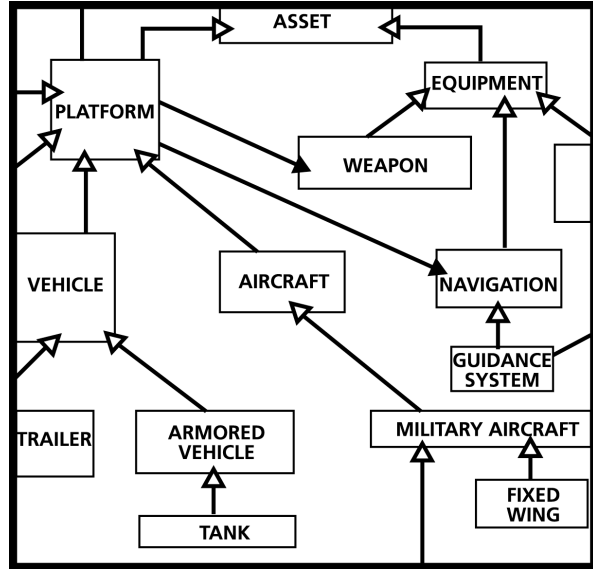


Figure 4.2: Small part of an ontology

Many years before computers became available this process of deductive reasoning was already well understood. Emil Post (1943) coined the term *productions* to describe sequences of IF...THEN conditions and conclusions. More familiar to the layperson is the term *rules*. The IF-part (or predicate) of a rule establishes the conditions that must be satisfied before the THEN-part (or consequent) can be assumed to logically follow. As shown in Figure 4.1, a single rule may contain multiple conditions and/or actions. In addition, secondary conditions can be embedded in both the IF-part and the THEN-part of a rule.

Rules are of course not the only way in which we can structure problem conditions and a solution sequence. For example, a neural network consisting of interconnected layers of input, intermediate and output nodes utilizes an entirely different approach for detecting a pattern of conditions. It essentially implements a sophisticated mathematical function to generate a very primitive numerical output (i.e., a set of decimal values between 0 and 1) to indicate that a particular input (represented in the same numerical manner) is similar to an input pattern that it has been mathematically trained to recognize. This is quite different from the approach that the rule shown in Figure 4.1 follows to logically define the conditions that must be met before any of the actions can take place. Instead, the neural network relies on a pattern matching approach that does not require an understanding of the meaning of the recognized patterns but simply the ability to recognize it. Furthermore, additional interpretation has to be provided by other means to convert the real world pattern into an abstract set

of numerical values that are fed into the input nodes and convert the numerical output code generated by the neural network into a meaningful (real world) result. Neural networks are powerful tools, even though they do not rely on symbolic reasoning capabilities.

Software agents that are able to analyze problem situations, dynamically changing conditions, and events, must have some understanding of the meaning of the information that they are reasoning about. The question then becomes: How can we create a computer-based environment that conveys to a software agent sufficient meaning for that agent to undertake reasoning tasks of the kind exemplified by the rule shown in Figure 4.1?

To answer this question it is necessary to reiterate the previous brief reference to the distinction between data and information (see Section 3.1). Data are simply numbers and words, while information adds to data additional very important concepts, abstractions and contextual *relationships*. The addition of concepts, abstractions and relationships is critical to any reasoning process because they provide *context*. Without this context even a human being would have great difficulty making sense out of a bunch of data. What makes it so easy for us human beings to reason about a wide range of data is the context that we have accumulated in our cognitive system over time through an experience-based learning process. We automatically convert data to information as long as we can find in our memory the context within which the words and numbers (i.e., data) that our eyes see, convey meaning. In other words, subject to the existence of relevant experience our cognitive system automatically adds the relationships (i.e., context) that are necessary for us to reason about the data.

Furthermore, it is through abstraction that we are able to apply our experience to a wide range of problems, even those that fall outside our specific experience base. This is partially due to the ability for these abstractions to essentially apply to concepts or entities yet to be incorporated. For example, abstracting the notion of *trackability* not only supports currently utilized entities but at the same time lays a solid foundation for incorporating additional entities that may also embody the property of being *trackable*.

Since these human cognitive processes are automatic, it is perhaps not unreasonable for us to forget that computers do not have this capability because they do not have an equivalent cognitive system. The same would apply if we were to ask a literate six-year old child to interpret the meaning of a typical printed, single-page agenda of a business meeting. Although the child may be able to readily read the agenda she is unable to make much sense of its contents because she has no prior experience of such meetings. In other words, the child lacks the context that is necessary for reasoning about the agenda.

For the computer to be able to support automatic reasoning capabilities we have to create a software environment that incorporates context. This can be achieved fairly easily by constructing an information model as a virtual representation of the real world context within which software agents are expected to apply their reasoning capabilities. Such an internal information model is referred to as an ontology. A small part of a typical example of such an ontology is shown in Figures 3.7 and 4.2. It describes the real world context in terms of objects with characteristics and relationships. For example, in a military command and control context such objects would include different kinds of weapons, a wide range of infrastructure objects, weather forecasts, friendly and enemy units, and even conceptual objects such as the notions of threat, planning, mobility, and readiness. Generally speaking, the more relationships among objects that are included in the ontology the more context is

provided by the ontology, and the more powerful (i.e., intelligent) the reasoning capabilities of the software agents are likely to be.

Without the context provided by an internal information model (i.e., ontology) there can be no meaningful, automatic reasoning by software agents. Of course there could still be neural network agents and software modules that simply manipulate data based on some predefined data-processing scheme, but neither of these are capable of the kind of symbolic reasoning that is now being referred to under the title of *intelligent agents*. Therefore, the missing link or essential prerequisite for intelligent agents is the existence of an internal information model that provides the necessary context for the symbolic reasoning activities of the agents. We human beings do not have to consciously invoke any action to relate what we see, hear and feel to the context held in our brain. The need for this context to be created in the computer is therefore not intuitively obvious to us. This is no doubt the principal reason why such a fundamental aspect of intelligent computer-based agents is still largely overlooked.

4.2 Planning and Re-Planning Functions

Planning is a reasoning activity about the resources and actions to fulfill a given task. Planning agents are complex agents that reason with the current environment and produce a plan based on the current state of objects and the current requirements. This planning process involves matching the requirements with the available resources and producing a course of action that will deliver the resources to the requesting objects. The complexity of the process can be reduced by breaking down the planning tasks among a set of agents. The basic tasks are:

- Identify the requirements.
- Identify the available resources.
- Report any shortage of resources for a given set of requirements.
- Identify the available set of actions.
- Generate a plan for fulfilling the requirements.

Plan generation is the actual planning activity in the above list of tasks. Many planning systems use specialized search algorithms to generate plans according to given criteria. Among the most popular search algorithms for planning is the A^* algorithm and its variations, such as iterative deepening IDA^* . The A^* algorithm finds the shortest path in a graph that represents desired goal states. Another general planning system is provided by the *Graphplan* methodology, which explicitly constructs and annotates a compact structure called a Planning Graph (Blum and Furst 1997). In this methodology a plan is considered to be a kind of *flow* of truth-values through the graph. This graph has the property that useful information for constraining search can be quickly propagated through the graph, as it is being built. *Graphplan* then exploits this information in the search for a plan.

Re-planning, which is also commonly referred to as continual planning, involves the re-evaluation of parts of an existing plan because of a change in the information that has been used in the creation of that plan. This is a common situation in real life planning. As soon as a plan has been created there is a strong possibility that the underlying information on which the plan has been based will change. Naturally, such information changes will affect the plan to varying degrees, and if sufficiently severe, will require the plan to be either modified or replaced with a new plan.

4.3 Truth Maintenance Approach to Re-Planning

In a real life system there is also a need to monitor the execution of the plan and make adjustments as needed. The adjustments to the plan are made by a truth maintenance agent. The truth maintenance agent monitors all the objects involved in a plan and modifies the plan according to changes in any of these objects.

Software agents that employ reasoning techniques, such as pattern matching or constraint-based reasoning, provide a mechanism for setting up conditions of interest and trigger actions when these conditions are satisfied. This paradigm offers an environment for truth maintenance in which states of objects are monitored and whenever the state of an object (which is a component of a plan) violates the set conditions the plan actions that rely on this object are invalidated and the requirements of these actions are sent back for re-planning.

The truth maintenance approach allows partial re-planning instead of complete re-planning. In large systems, a plan may involve several hundred (or thousand) of actions, each of which involves a large number of objects. The complexity is also increased by objects that participate in multiple action sets. In such systems, the complete re-generation of a plan may be costly and less helpful to the end-user. The change in object states usually takes place during the execution of a plan. It is often not feasible at this point during system operations to re-generate a complete plan, since some of the plan actions will have already been executed. Partial re-planning through truth maintenance allows the system to focus only on those parts of the plan that are affected by the change in object states.

4.4 Learning as Part of a Planning System

Some planning systems take advantage of the feedback obtained from the monitoring and execution of plans to add to their knowledge by employing learning techniques, such as explanation-based learning, partial evaluation, experimentation, automatic abstraction, mixed-initiative planning, and case-based reasoning. Such agents learn to appropriately coordinate their activities in order to optimally share resources and fulfill their own interest. There are many approaches to learning in agents, including reinforcement learning, classifier systems, and isolated concurrent learning. Learning techniques also enhance the communication ability of agents (Sen et al. 1994, Veloso et al. 1995).

4.5 Service Agents

Agents that are designed to be knowledgeable in a narrow domain, and perform planning or assessment tasks for other agents (i.e., human agents and software agents) are often referred to as Service Agents. They are endowed with a communication facility that allows them to receive and send information. The manner in which they participate in the decision making activities depends on the nature of the application. Service Agents can be designed to respond to changes in the problem state spontaneously, through their ability to monitor information changes and respond opportunistically, or information may be passed to them in some chronological order based on time-stamped events or predefined priorities. They should be able to generate queries dynamically and access databases automatically whenever the need arises. In other words, Service Agents should

have the same data search initiation capabilities as the user and should not be dependent solely on the user for access to external information sources. In fact, the human users in such multi-agent systems may be categorized as very intelligent, multi-domain service agents. Examples of such Service Agent systems can be found in the literature (Durfee 1988, Lesser 1995, Pohl et al. 1989, 1991 and 1997).

Within a networked environment the Service Agents pertaining to a single multi-agent system or service may be distributed over several computers. Alternatively, several single multi-agent services can collaborate. In this case each multi-agent service functions as an agent in a higher level multi-agent service. Such services are well suited to planning and re-planning functions in which resources and viewpoints from several organizational entities must be coordinated. The operator at each node should be able to plan in multiple worlds. For example, a private world in which shared information sources may be accessed but the deliberations of the operator are not shared with other operators, and a shared world which allows and encourages the continuous exchange of comments, plans and instructions. The capability normally exists for the operator to maintain multiple views of each world to facilitate experimentation and the exploration of alternatives. The Service Agents resident in each system (i.e., at each node) should be able to differentiate between worlds and also between the views of any particular world.

4.6 Mentor Agents

A Mentor Agent is an agent type based on the agentification of the information objects that are intrinsic to the nature of each application. These are the information objects that human decision makers reason about, and that constitute the building blocks of the internal representation (i.e., ontology) of the problem situation within an information-centric application. The concept of Mentor Agents brings several potential benefits.

First, it increases the granularity of the active participants in the decision making environment. As agents with communication capabilities, objects such as an *unmanned vehicle*, manned vehicles, any individual supply items, or even requisition orders, can pursue their own needs and perform a great deal of local problem solving without continuously impacting the communication and coordination facilities utilized by the higher level components of the decision-support system. Typically, a Mentor Agent is a process (i.e., program) or component of a process that includes several adjuncts that provide the agent with communication capabilities, process management capabilities, information about its own nature, global objectives, and some focused problem solving tools. Second, the ability of Mentor Agents to request services through their communication facilities greatly increases the potential for concurrent activities. Multiple Mentor Agents can request the same or different services simultaneously. If necessary, Service Agents responding to multiple service requests can temporarily clone themselves so that the requests can be processed in parallel. Third, groups of Mentor Agents can negotiate among themselves in the case of matters that do not directly affect other higher level components or as a means of developing alternatives for consideration by higher level components. Fourth, by virtue of their communication facilities Mentor Agents are able to maintain their associations to other objects. In this respect they are the product of *decentralization* rather than *decomposition*. In other words, the concept of Mentor Agents overcomes one of the most serious deficiencies of the rationalistic approach to problem solving; namely, the dilution and loss of relationships that occurs when a complex problem is decomposed into sub-problems. In fact, the relationships are greatly strengthened because they

become active communication channels that can be dynamically created and terminated in response to the changing state of the problem situation.

Mentor Agents may represent abstract concepts such as mobility or threat, collective notions such as climate, virtual entities such as a simulated supply route during a planning process, and physical objects such as an *unmanned vehicle* in the battlefield. For example, consider a small communication device that is embedded in a computer tag and attached to a UAV. This Radio Frequency Tag (RF-Tag) is capable of receiving and sending messages to a Mentor Agent taking the role of an expediter within the computer-based command and control system. In a tactical *unmanned vehicle* scenario the Mentor Agent can serve many functions. It can seek targets based on automatic detection of specific terrain patterns, generate warnings and alerts relating to hazardous atmospheric conditions, maintain contact and collaborate with other UAV, UVG or UMS *unmanned vehicles* in combined operations, and so on.

5. Web Services

As discussed previously (see Section 3.2) the phrase Service-Oriented Architecture (SOA) refers to a design concept for distributed systems that emphasizes loose coupling and reuse (McGovern et al. 2003). In a SOA-based information management environment the primary building block is the *service*. The notion that a service is *discoverable* means that each service can describe itself publicly in a way that enables other software entities to learn about its existence and to determine whether that service is useful to them. This is a very important building block for constructing loosely-coupled, extensible systems. *Registration* is the act of describing a service in a standardized, publicly accessible manner. *Discovery* is the other side of the equation, namely: once a service has registered itself, would-be clients search registered services for ones that suit their purposes. Finally, *access* is the result of a client discovering a service, namely: the client *accesses* the service using information gained from the discovery process.

5.1 Web Services as an Implementation of the SOA Concept

The term *Web service* refers to an implementation of a SOA-based environment that uses standard protocols from the World Wide Web as the underlying communication mechanism. This concept allows Web site operators to extend the value of their offerings by enabling software consumption of the same information that the site provides for human use.

On an intranet level, enterprises are finding that integration of existing systems can be simplified by first constructing specialized Web services that expose the functionality of these systems for programmatic access using open standards. Integration of these systems then reduces to the construction of Web service clients to access the services. This has proven to be both simpler and more flexible, in many cases, than previous approaches to integration.

The base-level technology for Web services is the eXtensible Markup Language (XML). All specifications are XML-based, and communication in a Web services environment is generally assumed to consist of XML documents exchanged among all participants. The use of XML is intended to promote loose coupling and to avoid reliance on proprietary technologies. However, it is widely understood that there may be situations where the large size of XML files, when compared with their binary equivalents, may prove problematic for performance and bandwidth reasons. As a result, the specification of Web service access allows for the inclusion of other types of communication such as the Web Service Description Language (WSDL).

One of the key goals of all Web services standards is to enable future extensibility to accommodate new technologies and ideas for SOA-based information management environments. This goal ensures that the use of current standards will not prove to be a dead end, but will continue to be viable over time. The addition of specific extension points into specifications such as UDDI (Universal Description, Discovery and Integration) and WSDL also allows for flexibility in the deployment and use of Web services using today's technologies.

5.2 Simple Object Access Protocol (SOAP)

The standard for all communications among Web services and consumers is the Simple Object Access Protocol (SOAP). SOAP defines an envelope that wraps around the content of the

communication and provides both metadata and other information such as routing, which are separate from the content of the communication.

5.3 Universal Description, Discovery and Integration (UDDI)

In Web service implementations of the SOA concept, the aspects of registration and discovery are handled by registries implementing the Universal Description, Discovery, and Integration (UDDI) standard. UDDI defines a set of XML schemas that can be used to construct documents describing types of services, types of business organizations, and individual service descriptions. UDDI also defines XML-based statements for adding these descriptions to a UDDI-compliant registry, as well as queries that enable Web service consumers to locate services that fill their requirements.

5.4 Web Services Description Language (WSDL)

While UDDI allows Web service providers to register services and businesses by type, it does not itself define a means of describing how a service consumer can access a given service. This is left to other standards, adding yet another vantage point for future extensibility. The current means of describing the location and parameters of a given instance of a service type is the Web Services Description Language (WSDL). Each service entry in a UDDI registry may include a reference to a WSDL-based description of the details of accessing that service. Each WSDL document can define one or more access points (i.e., service locations) along with the schema for the parameters accepted by the service at that access point, and the return value type.

Together, UDDI and WSDL allow service consumers to locate and access specific service instances, and to ascertain what kinds of SOAP documents each service will accept. Some of this information can be utilized at run-time (e.g., the access points) but most of it is primarily useful when the service consumer is being designed and implemented. At that time the information from the registry can be used by developers to determine what the consumer will be required to send to and receive from each type of service.

5.5 Federated Web Services

Many of today's Web services are fairly simple in terms of information flow. Typically, a consumer contacts a service with a request for some action to be performed. The service performs the action and returns the result. This style of service works well in simple scenarios, but does not take advantage of many of the more advanced possibilities inherent in the concept of a SOA. As more Web services become available, more complex architectures are beginning to emerge. In particular, Web services are themselves becoming consumers of other Web services, so that any given service can be seen as a building block that provides part of the functionality of other services.

Joining the functionality of multiple services together and presenting the result as a single service is referred to as a *composite* or *federated* service. This approach allows the rapid creation of larger services using existing services, while keeping consumers simple, since they are not required to connect to each individual service in order to achieve the desired results. Federated services, however, create problems that do not exist in the simpler case. Many of these problems are related to security issues.

5.6 Web Services Security

One of the most important considerations for Web services is the question of secure communications (Siddiqui 2004). When an organization deploys a service that implements some functionality for clients across the Internet, there is always the need to ensure that the output from the service will be received only by authorized clients, and that the incoming request has not been altered at any point between a client and the service. Clients, on the other hand, need to be assured that they are connected to an authentic server, and that the response that they receive has not been intercepted and modified as it is passed back from the server.

Web service security is one of the basic prerequisites for the adoption of this technology on an Internet-wide basis. Without adequate security, Web service deployment is likely to continue at an enterprise scale, but no further. Both the Organization for the Advancement of Structured Information Standards (OASIS) and the World Wide Web Consortium (W3C) are engaged in ongoing work to develop specifications for various aspects of Web services security. Many of these specifications have now been finalized and have multiple implementations.

There are many standards, which work together to provide various aspects of Web service security. These standards can be applied as needed. However, they do not constitute a single monolithic block, but provide a flexible framework that enables individual service providers to determine the level of security they need (CDM 2004).

6. Interoperability Bridges

There is a critical need for an information-centric software environment to be able to support interoperability among heterogeneous external systems. However, the desired interoperability must go beyond the elementary communication of data and endeavor to support a more powerful context-oriented relationship among systems. An important consideration in such functionality is the support, or rather the promotion, of meaningful interoperability while still retaining individual system representations, or perspectives. In other words, the meaningful integration of potentially disparate systems in a manner that allows the collaborating systems to retain their unique means of representing, or perceiving, the domains over which they operate. Existing approaches to this problem include the provision of a specific translator for each source/target system pair combination, development of a universal ontology to encompass both systems, and ontology facades that specialize from a broad representation to a narrower perspective. Writing a specific program is not a trivial task, requiring detailed knowledge of both systems and does not lend itself well to changes in either system. Developing a universal ontology is not only highly impractical but also requires an ongoing effort of significant proportions to achieve an even remotely acceptable ontology.

To avoid the potential complexity inherent in mapping between highly disparate perspectives it is likely that a more suitable solution will require the employment of reasoning-enabling technologies capable of supporting the complex analysis involved in performing context-based translation. This approach appears to be particularly amenable to a service-oriented model of inter-system collaboration. In this paradigm, each collaborating system is essentially defined as a collection of available services. Interoperability in this sense involves systems employing each other's services in an effort to perform their desired tasks.

6.1 Problem Definition

As the demand for sharing information increases, an additional burden is placed on the tools and systems that support the decision-making process. Context-oriented systems, as opposed to data-centric systems, rely heavily on the contextual depth of the descriptions over which they operate. Contextual depth, or semantics, forms the fundamental enabler for such systems to offer users helpful assistance in the decision making process. Driven by the need for systems to understand more about the problems they are helping to solve is the need for such systems to interoperate. Whether interacting with other context-enabled systems or accepting data feeds from legacy data-centric systems, the need to understand the semantics of what is being passed places a significantly higher burden on the representational depth of the overall exchange (Karsai 2000).

Among the multitude of issues surrounding the subject of meaningful interoperability, it is the fundamental strength of context-centric, decision-support systems that poses the most challenging problem to this endeavor. This critical enabler, and simultaneous nemesis, is representational depth. As the name implies, the context-oriented approach to building decision-support systems endeavors to go beyond the data-centric approach to representation (i.e., isolated chunks of typically numeric or string-based data with little or no inter-relationships and essentially void of any embedded semantics) and incorporate the potentially numerous relationships, implications, and rules that are needed for the more complex analysis inherent in agent-based, decision-support systems. A critical aspect of such representational depth is perspective. The biases associated with how something is

viewed are very significant to the decision-making process within a particular domain. As such, perspective is a critical ingredient to effective context-oriented representation. Supporting the perspective of viewing a UAV as a sequenced collection of assembly stages may be more appropriate, and effective, to an assembly-line management decision-support system than to view the UAV as a means of performing surveillance missions, the latter being more appropriate for a tactical command and control system. The converse to representational depth is the increased disparity that inevitably develops between the representations upon which particular systems operate. In other words, the most empowering ingredient in context-oriented computing also presents one of the most difficult issues to deal with when such systems are required to interoperate (Pohl 2001, Pohl et al. 1997).

As a result for any meaningful interaction to occur between context-oriented systems there is a need for a translational component. To preserve the native biases (i.e., perspectives) inherent in each interacting system, exchanged context must be transformed in a manner that incorporates the applicable perspectives of the receiver. This requirement can be satisfied with a Web service type, translational interoperability bridge.

6.2 Solution Criteria

To successfully address the issues presented above, a solution is required to meet several criteria, including compliance with available standards, flexibility, and reusability. One of the primary goals of any solution intended for repeated application to varying interoperability scenarios is the ability to be flexible. A key aspect to such adaptability is the clear separation of framework from application specifics. In other words, support for the various abstractions associated with translation-based interoperability should be designed and implemented as a reusable framework. This framework should also identify the necessary interfaces outlining its connection to the application-specific side of the equation. The latter requiring the necessary functionality to effectively *adapt* client systems to the particular interoperability framework and interaction model presented by the solution.

Another important quality of a candidate solution is the promotion of available industry standards. This is particularly significant when a high degree of reusability by numerous parties is intended. Accordingly, the application of such a tool should center on industry-familiar technologies, standards, and tools. Not only does adherence to available standards aid in adoption of a particular solution but it constitutes an endeavor of significant importance in a field where complexity and one-off solutions abound.

As mentioned previously, a critical consideration in the quest for interoperability among information-centric systems is to maintain the individual perspective of each system. The logic required to support translation between these perspectives presupposes a level of reasoning akin to expert systems. In many respects, for the more complex context-oriented interactions, a level of decision-support on par with solutions supporting multi-variable, complex problems may be appropriate. More often than not this will include human decision makers to represent higher-level concepts not able to be adequately represented in software with current technology.

An additional, often times overlooked, criterion for a successful solution to the interoperability problem is the need to support a more straightforward transformation without incurring the

overhead associated with complex context translations. The goal is to support a range of complexities and to limit any incurred overhead to situations where it cannot be avoided.

A promising solution takes the form of a Web service, interoperability bridge based on a reusable framework. The interoperability bridge enhances the traditional Web services architecture by also addressing the issue of differing representations between service users and service providers. As indicated earlier, supporting meaningful interaction among potentially disparate perspectives and subsequent representations is particularly important when dealing with context-oriented systems.

The application of the SOA-based *service request* metaphor to inter-system collaboration allows each interoperating system to view other systems, and expose itself, as a collection of available services. The resulting interoperability model promotes a decoupled environment requiring no notion of system identity other than the standard service descriptors registered with the bridge's Web services registry. Further, due to the embedded translational quality of this solution, bridge clients (i.e., service users and service providers) need not be concerned that the other might *speak a different language*. In support of such an interoperability model a number of existing and emerging technologies may be employed.

6.3 Technologies Employed

The proposed Web service-based, translational interoperability bridge incorporates a number of prominent technologies to accomplish its goal. First among these is a Web services architecture (Daconta et al. 2003, Ewalt 2002). By supporting standardized service lookup registries and interaction protocols, Web services architectures present an extensible decoupled, capability-oriented model for system interaction. In this model services are employed on an as-needed basis allowing the classical notion of operational boundaries to effectively expand and contract, as additional capabilities are needed. Furthermore, adhering to standard interaction protocols, systems are empowered with a vehicle for discovering and engaging new capabilities. Although the issue of semantic discovery is still an area of significant research, SOA-based Web service environments lay an effective foundation for such discovery-oriented dynamics.

XML (Gil and Ratnakar 2002) together with its XSLT language counterpart (Cagle et al. 2001) are two additional technologies employed by interoperability bridges. XML provides a flexible means of defining structure through the use of syntactical tags. XML schemas can be developed to describe the structural aspects of entities, notions, and concepts. Receivers can process XML documents based on these schemas in an interpretational manner. The result is a means whereby software components can process incoming content based on a previously unknown representation. However, it should be noted that such discovery is limited to structural characteristics and does not include the discovery of semantics or *meaning*, which is vital in information-centric decision-support systems. Even considering a schema describing the domain of concepts, rules, and implications, there is still a requirement for a pre-defined understanding by the receiver of the basic concepts, rules, and implications of the domain. At some point, the semantics need to be adequately represented beyond simply their structure. While structural discovery plays a significant role in the eventual goal of true contextual discovery, it is only a piece of the puzzle.

The ability to describe discrete, interpretable structure can be exploited to support structural transformation between XML schemas. XSLT is one such language that can be used to describe exactly how content based on one schema can be mapped to another. XSL transforms, or rules, can

be defined statically or dynamically and can be effectively applied in the case of straightforward, property-to-property translation. However, for the complex transformation requirements of context-oriented representations a more powerful paradigm is required. The additional reasoning required for this level of transformation can be successfully addressed through the use of inference engine based technology. Similar to XSLT, inference engine based transformation represents transformation logic as sets of managed rules. However, in the case of inference engines, these rule sets can be significantly more complex with support for extensive condition-based pattern matching and the subsequent management of progressively satisfied pattern matches. Some examples of rule-based inference engines are the CLIPS expert system shell developed by NASA (NASA 1992) and the Java Expert System Shell (Jess) inference engine developed by Sandia Laboratories (Friedman-Hill 2003). In either case, complex transformation logic can be implemented as expert systems applying various levels of reasoning to determine the appropriate transformation.

7. UAV Image Interpretation: *An Example Demonstration*

Autonomous visual tracking of objects is a capability that can greatly enhance the autonomy level of unmanned vehicles operating in the air as well as defense stations located on the ground. Tracking involves the task of following a moving object within a video sequence. There are several challenges to autonomous object tracking, including: object lighting changes; color and shape variations; complex and cluttered background scenery; temporary partial occlusions; objects temporarily disappearing from the scene; and, changes in scale of the tracked object. While there are still no trackers currently available that are able to match human capabilities, significant progress has nevertheless been made in the effectiveness of tracking algorithms.

The authors have implemented a tracking demonstration that utilizes an algorithm to track objects in a video sequence utilizing an online learning algorithm. The underlying technology of this tracking algorithm and its application, as well as future research suggestions are described in this section of the report.

7.1 Object Tracking

Object tracking in video sequences involves the task of following a particular object in a video stream. There are many tracking algorithms that have been presented in the literature. For this demonstration application, we have chosen to build upon a machine vision tracking algorithm presented by Grabner and Bischof (2006). This algorithm utilizes the online Adaboost technique to build a highly accurate classification system. The online nature of the algorithm allows the set of tracking features to be dynamically adapted to the varying conditions present in the video and in the tracked object. Due to the dynamic nature of the algorithm, tracking is often remains correct even in the presence of changes in visual appearance of the tracked object.

7.2 The Adaboost Algorithm

Adaboost is short for *Adaptive Boosting*, which is a machine-learning technique used to improve the performance of several other machine-learning tools. By combining several poorly performing classifiers in a meaningful manner, Adaboost is able to merge the knowledge gained from each classifier into a single, highly-accurate classifier.

Weak Classifier: A weak classifier is one that performs slightly better than an educated guess of the correct answer. It is essentially a rule of thumb that they provides a general suggestion of what the correct classification answer is, given a set of input data. Such classifiers are referred to as *weak* because in general their accuracy is slightly better than unsubstantiated selection. The benefit of using weak classifiers is that the amount of computation necessary for training and use is usually quite modest, allowing them to operate very quickly.

Strong Classifier: A strong classifier is typically a linear combination of several weak classifiers. It is highly accurate and often represents the final classifier used in a classification task. Intuitively, it is based on the concept that the combination several rules of thumb can result in a classifier that is significantly superior to any of the individual weak classifiers. A strong classifier consists of the n weak classifiers and n weights (i.e., one

weight (w) for each of the classifiers). Each weight is a numeric representation of how much to trust the prediction of the particular weak classifier. If the output of each weak classifier is either -1 or $+1$ depending on the classification result, then the output of the strong classifier can be computed in the following manner:

$$\text{strong classifier output} = w_1 \times h_1^{\text{weak}}(\mathbf{x}) + w_2 \times h_2^{\text{weak}}(\mathbf{x}) + \dots + w_n \times h_n^{\text{weak}}(\mathbf{x})$$

where: $h_n^{\text{weak}}(x)$ is the classification prediction of the particular weak classifier n .

A key procedure in the Adaboost algorithm is the computation of the relative weights of each of the weak classifiers. In the standard Adaboost methodology, a set of training inputs and outputs are used to train the classifier before the classifier is used to make any new predictions. As the training proceeds to compute the weights, the example inputs that are difficult to classify are given higher weights. This forces the algorithm to focus on training examples that are difficult to classify, rather than the ones that are easier to classify.

Adaboost has proven its effectiveness in classification tasks that have previously required a significant amount of computation using other machine-learning techniques. One example is in the area of face detection in still images, where implementations of face detectors based on boosted classifiers can perform detection very rapidly (Viola and Jones 2001).

The training process of the Adaboost algorithm typically requires the collection of several hundred positive examples and several thousand negative examples. The positive examples are samples of what the algorithm is attempting to detect. In the case of aircraft detection, for example, these would be sample image patches of various types of aircraft in various orientations. Having more positive examples will generally provide better accuracy in detection. Negative examples are typically random image patches that do not depict the desired detection object.

One major difficulty in this training process is the collection of the training data itself. It can be a tedious process to collect several hundred positive examples of a particular aircraft or object. Additionally and probably a more significant drawback to using Adaboost for tracking is that once a strong classifier has been created using several weak classifiers, the strong classifier cannot be easily modified to detect new objects without again going through the training process to create an entirely new classifier. This second drawback is particularly troublesome when using a classifier for tracking because a tracked object can have a greatly varying appearance during tracking. Successfully tracking an object requires an algorithm that can adapt to the lighting and appearance of an object. For these reasons an adaptive algorithm is a preferred approach. The online Adaboost algorithm described in the next section allows for adaptation based on object appearance.

7.3 Online Adaboost

Online Adaboost is a variation of the original Adaboost algorithm with the added ability to dynamically change which weak classifiers are used to create the strong classifier. Online machine-learning algorithms are those that adapt according to their internal parameters as more training data become available. As the algorithm is presented with more data during usage, the algorithm is able to add this new data to the learning algorithm.

In our work, we utilize the online Adaboost algorithm presented by Grabner and Bischof (2006, 260-7) as a basis for creating a strong classifier from several weak classifiers. In the online version of the algorithm, selectors are assigned to the visual features that most accurately distinguish the

object from the background. As the algorithm proceeds, whichever features correctly select the tracked object are given a higher weighting and thus their contribution to the overall prediction is increased. Those features that have been incorrect most recently have their weights reduced (or are completely removed from the current predictor) and thus their contribution does not greatly affect the predictor.

In the implementation of the online Adaboost algorithm, the amount of computation for each feature needs to be relatively small because the features are updated and computed several times during the search for an object in each frame. For our demonstration we have limited the types of features to those that can be computed very quickly, using known optimization methods. These features, which serve as the criteria for determining whether an image patch contains the object of interest or not, are described in the next section

7.4 Features for Tracking

Several different types of features can be used to track an object in a video stream. Aspects such as color, shape, and size give general descriptions of how a particular object appears. Various tracking algorithms have been designed about a particular feature or combination of visual features. Four different categories of features for tracking are employed in the demonstration system, namely: Haar-like features; gradient histograms; local binary patterns; and, average color.

Haar-like features compute the difference in intensity between two to four rectangular regions of an image. Example Haar-like features are shown in Figure 7.1 (a, b, c, d), such as the vertical Haar-like feature (Figure 7.1 (a)). A Haar-like feature is computed as the difference between the sum of the pixel intensities in the white rectangular regions and the sum of the pixel intensities in the black regions.

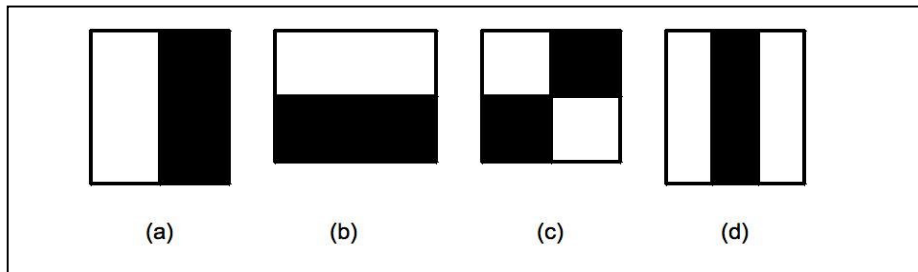


Figure 7.1: Typical Haar-like features

A vertical Haar-like feature will produce a strong response when there is an area in the image where there is a strong vertical edge. These features can be very small (e.g., 6 pixels by 6 pixels) and, in fact, there can be thousands of features within a small image area. As the dynamic Adaboost algorithm operates, it selects the Haar-like features that are most effective for differentiating the object from the background.

Haar-like features have recently proven quite popular for application in object detection tasks. For example, face detection is a task that has been successfully demonstrated using algorithms based on Haar-like features (Viola and Jones 2001). The Haar-like features are popular because they can be rapidly computed and are very simple in nature, yet they provide useful information when several are used in combination.

The second type of feature used in tracking is a gradient histogram. This feature is a histogram of the gradients (i.e., a metric of edge strength) over all the pixels within a search area. Prior work has shown that histograms of gradients provide the capability to accurately represent the shape of various objects (Levi and Weiss 2004). Intuitively, one can assume that an object that is tall and narrow will have more vertical edges than horizontal edges. The gradient histogram is represented by dividing the range of gradients into eight different directions. The gradient for a given pixel will fall into one of these eight bins. The contribution to the histogram bin is then provided by the magnitude of the particular edge. This process is computed for all of the pixels within a given search area. Objects that have a similar contour shape or internal edge distributions will also have gradient histograms that are similar.

The third feature used in the demonstration system is a *neighborhood* version of local binary patterns. Local binary patterns are a feature that has been shown to robustly detect various textures at different scales, as well as rotations (Ojala et al. 2002). The version of local binary patterns used in the demonstration system is a simplified 4-neighborhood version of the full 8-neighborhood feature. This simplified version allows for reduced computation to provide the additional texture information capability of local binary patterns.

The average color of a particular region is the final feature used in the demonstration system. When an object appears in a video sequence, the object will typically have a steady or slowly varying average color from one frame to the next. The nearby background region (i.e., the area surrounding the object) will tend to have a steady average color as well. In order to effectively capture this property, the average color of the pixels within a particular region of interest has to be computed. Similar to the other three features, the area of interest is very small and is always a small percentage of the total area being tracked. The average color is computed by averaging the red, green, and blue components of the pixels. Accordingly, each average color feature consists of three color components values.

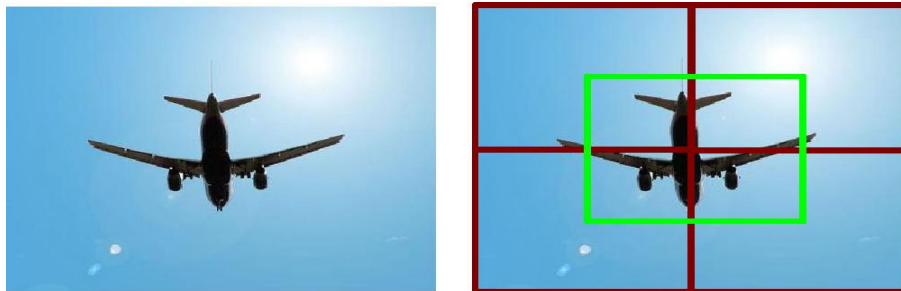


Figure 7.2: Determination of positive and negative image values

7.5 Tracking as Classification

In the demonstration system the task of tracking an object is treated as a binary classification problem. A particular area of the image can be classified as either containing the object of interest or not containing it. By approaching the tracking problem in this manner we can utilize an Adaboost-based classification algorithm. This approach has been confirmed by the work of Grabner and Bischof (2006, 260-7) and allows the tracking algorithm to be trained as a binary classifier by giving it positive and negative examples. Positive examples are regions of the image that are believed to contain the object of interest. Negative examples are regions of the image that do not

contain areas of interest. In reference to Figure 7.2, for a given frame of video there are one positive example and four negative examples. The selection of the positive and negative examples is shown in the image on the right (Figure 7.2). The positive example is within the green area of the image where the algorithm believes the object exists. The four negative examples are within the four nearby regions that are centered on the four corners of the positive example. This gives the four nearby regions values that partially contain both the foreground image and the background image.

For a given frame, a set of features (i.e., 150 in total) is computed for the positive and negative samples. Each feature computation provides a value or set of values that represent the strength of that feature in an area of the image. The resulting data from each of the features are used to train a simple Weak Classifier to distinguish between the positive and negative samples. Each of these features is considered a Weak Classifier because by itself it cannot robustly determine whether or not the box contains the object to be tracked. However, when several features are used in combination the results are quite robust.

7.6 Detecting the Object of Interest

Once the algorithm has been updated with one new positive example and four negative examples, the next step is to determine the most likely location of the object. This is accomplished by exhaustively scanning all possible locations of the object within the search area defined by the four negative examples. In the demonstration system there is no scaling of the search area or of the size of the tracked object itself. The algorithm simply performs an exhaustive search of the search area with the confidence of object detection being the magnitude of the output of the online Adaboost classifier. When all points have been scanned, the object center is simply the pixel that produced the highest magnitude from the classifier. This new center is then used to update the algorithm in the subsequent video frame.

7.7 Applications to Unmanned Aerial Vehicles

The ability to track objects autonomously in video streams allows for several potential applications in the unmanned aerial vehicle (UAV) domain. Autonomous tracking will allow UAVs to detect targets on the ground while traveling in the air. Tracking will also allow aerial vehicles to detect other obstacles that may be flying in the air. For ground applications, tracking of incoming aerial vehicle is a desirable capability and one that can relieve human operators of a tedious task.

There is much on-going research in the area of visual object tracking. We have constructed a proof-of-concept demonstration system to serve as an example of some of the technology currently available and the research directions that we would like to pursue in a proposed Phase II follow-on effort.

8. Phase II Proposal

As a Phase II follow-on project it is proposed to develop a proof-of-concept software system that demonstrates the ability of a set of intelligent tools, which are seamlessly integrated within a SOA infrastructure, to: detect a moving object (e.g., a car) on the ground from a UAV-mounted video camera; lock onto the object; collect visual information about the object; apply AI-based methodologies to match the object with one or more databases containing similar object types; and, determine the level of certainty with which it is predicted that this object is a specific object in the database. The proof-of-concept demonstration system will utilize an existing video stream imported through a driver, as a surrogate for a camera mounted on an UAV. However, the object detection and analysis will occur in near real-time to simulate the deployment of the software either on the UAV or at a remote control station.

8.1 Objectives of the UVORIS Proof-of-Concept System

The product of Phase II will be a working model of an Unmanned Vehicle Object Recognition and Interpretation System (UVORIS), capable of demonstrating the ability of intelligent agent technology in combination with other selected AI methodologies to detect and interpret moving objects on the ground captured by a video camera mounted on a UAV flying at some height above the ground.

The objective of the UVORIS proof-of-concept system is to combine various AI technologies, each of which has reached a proven level of maturity in a particular application domain, into an integrated multi-layered, multi-tiered software system environment that can support the surveillance mission requirements of a UAS. This objective is based on the concept that the highly demanding, multifaceted nature of the surveillance requirements can be met only by a hybrid AI solution. The requirements of a UAS surveillance mission include at least the following capabilities:

- The ability to detect moving objects in the air and on the ground and select one or more of these external objects for further action (e.g., closer scrutiny, tracking, or sampling).
- The ability to either automatically lock onto a selected object based on some criteria or through human controller action (e.g., clicking onto an object with a mouse).
- The ability to track the locked-on object for a period of time for purposes of collecting visual information about the object.
- The ability to transform the collected visual object information to the extent necessary so that it can be combined with context information (if available) to serve as a set of object characteristics.
- The ability to compare the characteristics of the locked-on object with a large number (i.e., several thousand) of instances of a similar object type in one or more reference databases.
- The ability to identify with the appropriate certainty factors the best matches of the locked-on object in the instance database(s).

- The ability of the human controller, or the system automatically, to adjust the weighting of each object characteristic and thereby influence both the matching process and the calculation of the certainty factors.
- The ability of the human controller, or the system automatically, to add object characteristics on an experimental basis to explore hypothetical *as if* conditions. For example, the automatically collected visual information may include the direction and speed of the moving object but not the destination. By analyzing the context of the mission a hypothetical, specific destination may be added to the collected information to explore the influence of this additional information on the matching process.

As a final requirement, the system should be able to perform all of these capabilities in parallel within reasonable performance and response parameters. Due to the dynamic nature of most military mission environments acceptable performance is likely to be measured in seconds rather than minutes.

8.2 Technical Approach

Apart from the moving object detection approaches and methods, which are described in some detail in Section 7, the following AI methodologies will be considered as components of the software solution.

8.2.1 Case-Based Reasoning

Case-Based Reasoning (CBR) is a similarity assessment strategy that is based on a two-part reasoning process. First, similarity assessment techniques are employed to compare a new, current case against a case base of previously stored cases to find the most similar case instance in the case base. Second, this case instance serves as a vehicle for classifying and resolving the current case (Zang et al. 2008). CBR utilizes a variety of data classification methodologies, including: Case-Based Classification involving the processing of textual data in a manner that is tolerant of ambiguous naming terminology, as well as terminology differences due to conceptual and spelling variations; Information Gain techniques that rely on the weighting of selected features based on the principle that rare items provide more semantic value than commonly occurring items; Synonym Detection involving the automatic creation of synonym libraries in support of semantic equivalence determination; Abbreviation Resolution techniques that exploit the typically fixed letter order of abbreviations; and, Model-Based Reasoning for the coordination of the activities of multiple software agents and the combination of their inferences into composite semantic equivalence scores (i.e., certainty factors).

In UVORIS these case-based similarity technologies will be utilized to disambiguate semantically equivalent but literally different data elements found in multiple reference sources.

8.2.2 Ontology-Based Representation

An ontology is 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 other words, ontologies are a virtual representation of real world situations (i.e., *context*) expressed in terms of formal specifications of domain concepts and relationships that can be processed by rule-based software modules (i.e., agents) with or without human assistance.

UVORIS will include an ontology consisting of several domains to provide the necessary context within which agents will match locked-on objects with similar object instances in reference databases.

8.2.3 Rule-Based Agents

In an ontology-based decision-support system that utilizes agents, the reasoning process relies heavily on the rich representation of objects and their relationships provided by the ontology. This representation provides a context on which agent logic can operate. Multi-agent systems emerged from the field of distributed artificial intelligence as a problem-solving mechanism for complex problems. The field focuses on the construction of multi-agent systems, where a set of intelligent agents interact, while pursuing a set of goals. In most cases the ultimate goal of these systems is the generation of a plan to meet certain objectives. For example, in the UAS domain such plans could focus on the movement of the aircraft to avoid obstacles, to track other objects either in the air or on the ground, or to navigate to a particular target site.

Planning is a reasoning activity about the resources and actions to fulfill a given task. Planning agents are complex agents that reason with the current environment and produce a plan based on the current state of objects and the current requirements. The planning process involves matching the requirements with the available resources and producing a course of action that would deliver the resources to the requesting objects. The complexity of the process can be reduced by breaking down the planning tasks among a set of agents. The basic tasks include: establishment of the requirements; identification of the available resources; determination of any shortage of resources for a given set of requirements; identification of the available set of actions; and, generation of a plan for fulfilling the requirements.

Plan generation is the actual planning activity in this list of tasks. Many planning systems use specialized search algorithms to generate plans according to given criteria. Among the most popular search algorithms for planning is the *A** algorithm, which finds the shortest path in a graph that represents desired (i.e., goal) states. Another general planning system is the *Graphplan*, which explicitly constructs and annotates a compact structure called a Planning Graph, in which a plan is a kind of *flow* of truth-values through the graph. This graph has the property that useful information for constraining search can quickly be propagated through the graph as it is being built. Graphplan then exploits this information in the search for a plan.

Re-planning, which is also referred to as continual planning, involves the re-evaluation of parts of an existing plan because of a change in the information that is involved in the creation of that plan. This is a common situation in real life planning. As soon as a plan is created there is a possibility of information change which may affect the plan. Re-planning can be decidedly more complex than planning because the agent(s) must first decide whether to modify the existing plan or generate a new plan. If it is decided to modify the existing plan then the agent(s) must determine which parts

of the plan need to be modified and how that modification should be undertaken so as not to conflict with the remaining parts of the existing plan.

8.2.4 Statistical Object Recognition Methodologies

Recent advances in statistics modeling from the fields of neural networks and more generally statistical query networks have shown great promise in established problem domains such as object detection and classification. Statistical query networks fit into the artificial intelligence area of supervised learning.

Supervised learning with statistical networks typically relies on a three-stage process. The first stage requires a human engineer to create the structure of the network that will be used for a particular problem domain. In the second stage the network is trained on a large quantity of data (i.e., input and output examples) that provides a good statistical representative sample of the entire problem domain. Finally, the third stage provides the network with unforeseen input examples and allows it to approximate the matching output. These techniques have been a research area for some time dating back to the Perceptron invented by Frank Rosenblatt in 1957 (Rosenblatt 1958). However, statistical models fell into disfavor in 1969 when Marvin Minsky and Seymour Papert showed that it was not possible for the Perceptron to learn some mathematical functions (Minsky and Papert 1969). Several years later Stephen Grossberg demonstrated that with some modification to the Perceptron and new learning algorithms these limitations could be overcome. Unfortunately regardless of this finding the Minsky and Seymour paper hindered the development of neural network research for many years. From the 1980s onward there have been an increasing number of successes that have renewed interest in the field of statistical networks.

Geoffrey Hinton and his co-workers in Toronto, Canada have applied statistical query techniques, specifically using a restricted Boltzmann machine (RBM), to the problem of hand written digit recognition on the MNIST data set (Hinton 2007, Hinton and Salakhutdinov 2006, Hinton et al. 1992). This data set contains a large number of small images with a hand written digit on them and a label of the image (e.g., 2). To date, Hinton's RBM approach is one of the most successful applications for the MNIST data set and has therefore shown that neural networks are still a very active and potentially fruitful area of study for image interpretation and machine vision. The same group has applied this approach to the Netflix movie recommendation challenge with some success and has also begun work on facial recognition techniques (Hinton et al. 2006, Yee and Hinton 2000). With continued improvements like these it is likely that statistical query networks will become a valuable part of object recognition techniques for the unmanned vehicle domain in the near future.

8.2.5 Semantic Database Search Techniques

The scope of database query facilities desirable for the kind of multi-agent, data interpretation, information fusion and intelligence analysis environment proposed for the UVORIS prototype far exceed traditional database management system (DBMS) functions and query capabilities. They presuppose a level of embedded intelligence that has not been available in the past. Some of these desirable features include: conceptual searches instead of factual searches; automatically generated search strategies instead of predetermined search commands; multiple database access instead of single database access; analyzed search results instead of direct (i.e., raw) search results; and, automatic query generation instead of requested searches only.

A traditional DBMS typically supports only factual searches. In other words, users and applications must be able to define precisely and without ambiguity what data they require. In complex problem situations users rarely know exactly what information they require. Often they can define in only conceptual terms the kind of information that they are seeking. Also, they would like to be able to rely on the DBMS to automatically broaden the search with a view to *discovering* information. This suggests, in the first instance, that an intelligent DBMS should be able to formulate search strategies based on incomplete definitions. It should be able to infer, from rather vague information requests and its own knowledge of the requester and the problem context, a set of executable query procedures. To facilitate this process the DBMS should maintain a history of past information requests, the directed search protocols that it generated in response to those requests, and at least some measure of the relative success of the previous search operations.

A traditional DBMS normally provides access to only a single database. The knowledge-based environment proposed for the testbed will involve many information sources, housed in a heterogeneous mixture of distributed databases. Therefore, through the internal-level database representations the DBMS must be able to access multiple databases. Using the mapping functions that link these internal representations an intelligent DBMS should be capable of formulating the mechanisms required to retrieve the desired data from each source, even though the internal data structures of the sources may differ widely. Particularly when search results are derived from multiple sources and the query requests themselves are vague and conceptual in nature, there is a need for the retrieved information to be reviewed and evaluated before it is presented to the requester. This type of search response formulation facility has not been necessary in a traditional DBMS, where requesting users or applications are required to adhere to predetermined query protocols that are restricted to a single database. Finally, all of these capabilities (i.e., conceptual searches, dynamic query generation, multiple database access, and search response formulation) must be able to be initiated not only by a human user but also by any of the computer-based agents that are currently participating in the data-to-information transformation operations.

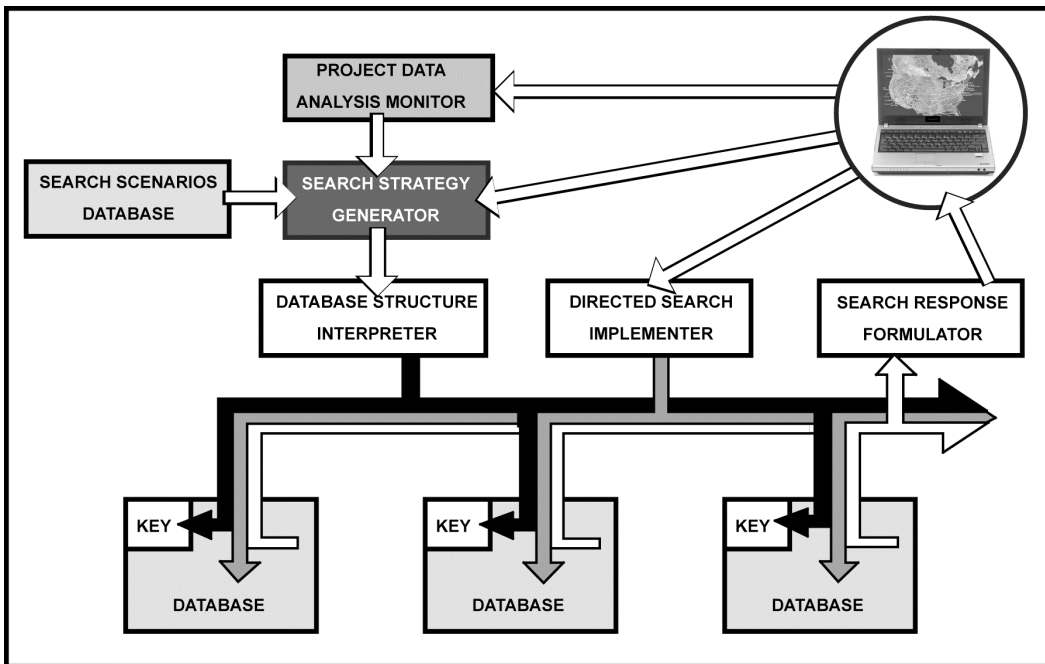


Figure 8.1: Conceptual model of a semantic search environment

A conceptual model of an intelligent search facility with the capabilities described above is shown in Figure 8.1. The proposed model is designed to support the following typical information search scenario that is envisaged for the integrated and distributed, collaborative, multi-agent environment proposed for UVORIS. Queries that are formulated either by the user or generated automatically by a computer-based agent are channeled to a Search Strategy Generator. The latter will query a Search Scenario Database to determine whether an appropriate search strategy already exists from a previous search. If not, a new search strategy is generated, and also stored in the Search Scenarios Database for future use. The search strategy is sent to the Database Structure Interpreter, which automatically formulates access protocols to all databases that will be involved in the proposed search. The required access and protocol information, together with the search strategy, are sent to the Directed Search Implementer, which conducts the required database searches. The results of the search are sent to a Research Response Formulator, where the raw search results are analyzed, evaluated and combined into an intelligent response to be returned to the originator of the query.

8.2.6 Knowledge Management Enterprise Services (KMES[®])

UVORIS will be based on SOA design concepts and implementation principles. It will incorporate an integrated set of functional capabilities in the form of intelligent web-based thin-client and distributed thick-client tools. These tools, referred to as Knowledge Management Enterprise Services (KMES[®]), are *self-contained* software modules with powerful functional capabilities and clearly defined interface specifications. They are designed to be platform independent and may be configured to any particular execution environment. However, most importantly they are *reusable* as components of any system that has a need for their capabilities. Some of these services may have quite narrow capabilities such as the mapping of data imported from an external source to an internal information model or ontology, while others will incorporate larger functional domains such as the optimum routing of groups of UAVs from multiple origins along alternative routes to multiple destinations.

The KMES[®] approach to software systems incorporates intelligent agent technology to analyze and categorize incoming signals and data, and then issue warnings and alerts as appropriate. The agents manipulate the incoming data within an internal information-centric representation framework (i.e., ontology) to publish statements of implication, and if so empowered, proceed to develop plans for appropriate action.

Existing data-centric systems lacking the adaptive, interoperability characteristics described above can be integrated into such an agent-empowered KMES[®] software environment through the use of *interoperability bridge* facilities that effectively map the data model in one system to the information model of the other. This allows for a meaningful bi-directional interaction and exchange of data in context. Such bridges have been successfully demonstrated by military organizations for linking legacy data-centric systems to intelligent command and control systems (Pohl et al. 2001). The technology is inherently scalable and allows for the efficient and effective interconnection of multiple participants within a heterogeneous net-centric environment.

The notion of *service-oriented* is represented as much in the elements of the three distinct tiers (i.e., information, logic, and presentation) of the proposed UVORIS architecture, as it is in the functional capabilities of each KMES[®] component. Therefore, even the internal elements of a KMES[®] component communicate through standard interfaces as they provide services to each other. They

are, in essence, decoupled software modules that can be replaced with improved modules as the technology advances. Each of these modules functions in an integrated fashion to form a comprehensive agent-based decision-support execution framework.

In summary, KMES[®] modules are intelligent, self-contained software components that are capable of performing narrowly defined tasks within a net-centric system environment. They are distinguished from legacy software systems in five ways. First, they adhere to clearly defined application programming interface (API) specifications, but are otherwise decoupled from the service requesters that they interface with. Second, they are reusable and can be interfaced with systems that require the kinds of services that they provide. Third, they are able to provide their services to both human users and other software systems. Fourth, their capabilities are enabled when they are configured (i.e., initialized) with the terminology and notions (i.e., ontology) of the target domain. Fifth, they are self-contained and can be replaced by improved modules as the technology advances.

The principal benefits of KMES[®]-based software systems are: *rapid delivery* of meaningful capabilities; *lower cost* due to the replacement of the normally prolonged software development period with a much shorter KMES[®] integration period; *greater reliability* and quality due to exhaustive core-component verification and validation in conjunction with the maturity that comes from extensive in-field use; *interoperability* through component design based on standard protocols and a decoupled, multi-tiered framework; *flexibility* to extend functional capabilities through plug-and-play components and an open architecture; and, *multiple deployment options* including net-centric delivery alternatives of hosted-managed services.

8.3 Phase II Tasks and Deliverables

With the objective of producing at the conclusion of the Phase II effort the UVORIS prototype as a working proof-of-concept system that will demonstrate the feasibility of applying AI technologies to significantly improve the surveillance capabilities of a UAS, the following tasks will need to be completed:

- Task 1:** Conduct knowledge acquisition with subject matter experts nominated by the Government to establish functional and technical requirements, including necessary interfaces to external systems.
- Task 2:** Prepare a functional and operational UVORIS Requirements Specification that includes system constraints and performance expectations.
- Task 3:** Employ normal software engineering processes to design the overall service-oriented architecture and prepare the more detailed technical specifications of the principal components and services, and their interfaces.
- Task 4:** Select the Services Management Framework components that will form the core of the SOA infrastructure (e.g., Enterprise Service Bus (ESB)). It is expected that as far as practical these will be Open Source software components.
- Task 5:** Conduct an experimental investigation of candidate object recognition methodologies and select those methodologies that are most appropriate for the

UVORIS prototype. It is expected that as far as practical these will be Open Source software components.

Task 6: Develop an ontology that will provide the necessary context for the object interpretation layer of the UVORIS prototype.

Task 7: Select, modify, and extend as necessary the existing reusable KMES[®] components that will be implemented as services within the UVORIS SOA-based infrastructure.

Task 8: Design and develop new KMES[®] components (if any) to provide necessary capabilities that cannot be provided by existing KMES[®] components.

Task 9: Prepare one or more video streams to serve as surrogate data feeds for the object recognition layer of the UVORIS prototype.

Task 10: Implement and test the selected object recognition methodologies utilizing the surrogate data feeds.

Task 11: Prepare one or more reference databases to provide the necessary instance data for matching objects recognized by the object recognition layer to object types and instances within a matched object type.

Task 12: Perform component integration, conduct unit testing of completed modules, and implement corrections for reported defects.

Task 13: Perform system integration, conduct comprehensive integrated testing, and implement corrections for reported defects.

Task 14: Prepare a UVORIS Technical Design Report, to include: functional and technical requirements; system architecture; component design of internal services; installation instructions; and, demonstration scenario description.

Task 15: Prepare presentation material for demonstration of the UVORIS prototype at a CONUS location and at such time as directed by the Government.

Task 16: Prepare a plan for the transition of the UVORIS prototype to initial operating capability (IOC), including: initial experimental deployment on a UAV; implementation of necessary software changes based on experimental results; deployment on an operational UAV for limited field testing; implementation of software changes based on user feedback; software testing; and, documentation.

- Deliverables:**
1. UVORIS Requirements Specification. [Due 2 months after award of contract]
 2. UVORIS Technical Design Report. [Due 6 months after award of contract]
 3. UVORIS Transition Plan. [Due 10 months after award of contract]
 4. Demonstration of UVORIS prototype at a meeting designated by the Government. [Due 10 months after award of contract]
 5. Software code on appropriate medium. [Due 10 months after award of contract]

8.4 Estimated Phase II Duration and Cost

The Phase II effort is projected to be of *10 months duration* at an estimated *cost of \$242,100*. Expected work hours within labor categories are listed below, together with travel and indirect cost estimates.

Labor Category	Rate	Hours	Cost
Program Manager	\$ 116.86	160	\$ 18,698
Project Coordinator Level 2	65.35	220	14,377
Software Engineer Level 1	57.22	1,240	70,953
Software Engineer Level 2	73.15	820	59,983
Software Developer, Level 1	37.12	1,240	46,029
Software Developer, Level 2	48.21	80	3,857
Systems Support Specialist Level 1	44.74	80	3,579
Systems Support Specialist Level 2	62.80	40	2,512
Technical Writer	37.55	120	4,506
Administrative Support, Level 2	50.88	120	6,106
		4,120	\$ 230,600
Travel:	6 person trips (@ \$2,000)		\$ 12,000
Materials and supplies:			\$ 100
		Total:	<u><u>\$ 242,100</u></u>

9. Acronyms and Definition of Terms

- ACR:** Area Coverage Rate
- ACTD:** Advanced Concept Technical Demonstration
- Agent:** This term has been applied very loosely in recent years. There are several different kinds of agents. Symbolic reasoning agents are most commonly associated with knowledge management systems. These agents may be described as software modules that are capable of reasoning about events (i.e., changes in data received from external sources or as the result of internal activities) within the context of the information contained in an internal information model (i.e., ontology). The agents collaborate with each other and the human users as they monitor, interpret, analyze, evaluate, and plan alternative courses of action.
- AI:** Artificial Intelligence
- C2:** Command and Control
- C2IEDM:** Command and Control Information Exchange Data Model
- CBRNE:** Chemical, Biological, Radiological, Nuclear, Explosive reconnaissance
- CLIPS:** C Language Integrated Production System
- COCOM:** Combatant Command
- COI:** Communities of Interest (COIs) are composed of groups that pursue common endeavors. Such groups typically share similar goals, objectives, problems, business processes, and vocabulary. Experience has shown that attempts to standardize nomenclature, logical data models, business rules, and processes at an enterprise level are usually counterproductive and eventually abandoned. Similar attempts to establish and enforce standards within smaller and more focused communities have been more successful, although not entirely without obstacles. Difficulties arise when COIs are not sufficiently specialized or when established standards are not adequately enforced within the community. DoD's Global Information Grid (GIG) vision assumes that with the explicit formal declaration of metadata it will be possible to automate the mapping of data and information that is required to be exchanged between any two COIs.
- CONPLAN:** Contingency Plan
- COSMOS:** Coalition Secure Management and Operations System
- COTS:** Commercial Of the Shelf software
- Data:** Strictly speaking data are numbers and words without relationships. Even though data are often stored in a relational database management system, typically only minimal relationships are stored with the data. Without adequate relationships, data do not contain sufficient context to support automatic reasoning capabilities by software agents.

Software that incorporates an internal representation of data (i.e., numbers and words) with few (if any) relationships is often referred to as *data-centric* software. Although the data may be represented as objects the scarcity of relationships, and therefore the absence of adequate context, inhibits the inclusion of meaningful and reliable automatic reasoning capabilities. Data-centric software, therefore, must largely rely on predefined solutions to predetermined problems, and has little (if any) scope for adapting to real world problems in near real-time. Communication between data-centric software applications is typically restricted to the passing of data-string messages from one application to the other. This imposes a larger transmission load than communication between information-centric applications. Since a data-centric application has no ‘understanding’ of the data that it is processing, a complete set of data must be transmitted so that the receiving application can process the transferred data in the appropriate predefined manner. For example, if the data to be transmitted involves the new location of an automobile then a complete set of data describing the automobile (including its new location) must be transmitted. In the case of information-centric applications only the new location and some object identifier would need to be transmitted, because both the transmitting and receiving applications have some ‘understanding’ of the general notion of an automobile and the specific instance of that notion representing the particular automobile that has changed its location.

Data Mining: Data Mining tools are applications that are designed to analyze the data in a Data Warehouse (or Data Mart) to establish relationships, identify trends, and predict future trends.

Data Portal: Typically, Data Portals are designed to provide access to operational data, with an emphasis on the presentation of data (usually to human users). Data Portals may also incorporate data analysis tools, and are often accessed in a Web-Services (e.g., Internet) environment. A Data Portal typically does not store data but provides a single point of entry to heterogeneous data sources.

Data Warehouse: Data Warehouses most commonly store and manage summarized (i.e., archived) data, usually in a relational database management system. However in some cases, for example when the operational data sources are particularly fragmented and unreliable, a Data Warehouse may serve as an operational data store that cleans and integrates data at the same granularity as they exist in the original data sources. The summarized or atomic data are periodically updated according to a predefined timeline. Data Warehouses often employ sophisticated data indexing mechanisms (e.g., based on key word indexing schemas) to facilitate the rapid retrieval of data. A subset of the data stored in a Data Warehouse that is focused on a particular functional area, is commonly referred to as a *Data Mart*.

DoD: United States Department of Defense

DSG: Distribution Steering Group

ESB: Enterprise Service Bus

FAA: Federal Aviation Administration

FCS: Future Combat Systems

GAO: United States Government Accountability Office

GES: Within the broad GIG vision, the GIG Enterprise Services (GES) will provide computing services that can be used by COIs to maintain their metadata and comply with enterprise-wide requirements in information management areas such as security, privacy, and application software installation and configuration policies. The GES computing services should also include software tools that will allow the automated verification of the compliance of ontologies with their underlying logical data models and standardized nomenclatures. Conversely, GES computing services could also assist application developers by generating a base set of ontology elements from the appropriate domain of a logical data model. The executive agent for GES in the DoD community is the Defense Information Systems Agency (DISA).

GIG: The Global Information Grid (GIG) is the vision that will guide the DoD in the implementation of an integrated network of knowledge management services and capabilities. Succinctly stated the GIG is envisioned as a net-centric environment of seamlessly interconnected data sources and utilization capabilities. This includes "... the globally interconnected, end-to-end set of information capabilities, associated processes, and personnel for collecting, processing, storing, disseminating, and managing information on demand to warfighters, defense policymakers, and support personnel." (Stenbit 2003, 1). The implementation of this vision will require: (1) the standardization of nomenclature and reference tables; (2) the definition of logical data models; (3) the publication of data encoding protocols and formats (i.e., metadata registries); (4) the adoption of data transmission standards; (5) the development of functionally created ontologies that allow data to be placed in context; (6) the publication of business rules and the encoding of these rules in software agents with automated reasoning capabilities; and (7) the formulation of policies, conventions, and processes, designed to facilitate planning, problem solving, and decision-making tasks.

All of these requirements are driven by the increasing need to automate at least the lower level data interpretation tasks that have been the almost exclusive province of the human users of computer systems. This has become necessary for several reasons. First, the increased ability to collect and store data in computers has created a bottleneck. The need to interpret the vast amounts of data has simply exceeded the availability of human resources. Second, human resources are desperately needed for higher-level information analysis to counteract increasing threats from

adversaries. Currently, most of the available human resources are fully employed at the lower levels of data interpretation. Third, there is an increasing need for more rapid and accurate decision-making capabilities. Typically, commanders and their staff find themselves in continuous re-planning mode as the most carefully laid plans require significant adjustments due to unforeseen events that will inevitably occur during implementation.

GS: Global Services

GWOT: Global War on Terrorism

HIS: Human Systems Integration

ICODES: Integrated Computerized Deployment System

IED: Improvised Explosive Device

IMMACCS: Integrated Marine Multi-Agent Command and Control System

IMO: Information Management Officer

Information: Information refers to the combination of data with relationships to provide adequate context for the interpretation of the data. The richer the relationships the greater the *context* (i.e., meaning conveyed by the combination of data with relationships), and the more opportunity for automatic reasoning by software agents.

Software that incorporates an internal information model (i.e., ontology) consisting of objects, their characteristics, and the relationships among those objects is often referred to as *information-centric* software. The information model is a virtual representation of the real world domain under consideration and is designed to provide adequate context for software agents (typically rule-based) to reason about the current state of the virtual environment. Since information-centric software has some 'understanding' of what it is processing it normally contains tools rather than predefined solutions to predetermined problems. These tools are commonly software agents that collaborate with each other and the human user(s) to develop solutions to problems in near real-time, as they occur. Communication between information-centric applications is greatly facilitated since only the changes in information need to be transmitted. This is made possible by the fact that the object, its characteristics and its relationships are already known by the receiving application.

IPL: Integrated Priority List

JC3IEDM: Joint Command, Control and Consultation Information Exchange Data Model

Knowledge: The term knowledge is commonly used to refer to the body of data, relationships, identified patterns, principles, rules, problem solving methods, and prototype solutions that are known in any domain. It may be explicitly recorded and described in text (e.g., books, journals, minutes of

meetings, etc.) or implicitly held in the cognitive system of individuals. Knowledge is typically acquired through experimentation, observation, analysis, and experience by human beings. The principal purpose of *knowledge management* is to make knowledge explicit and accessible throughout an organization, so that: (1) groups within the organization can effectively collaborate in the performance of tasks; (2) new members of the organization can rapidly acquire the necessary knowledge to become productive; (3) lower level information analysis and reasoning tasks can be automated; and, (4) past decisions, solutions and lessons learned can be captured and considered in future tasks.

LTA: Limited Technical Assessment

MEMS: Microelectromechanical Systems

Metadata: Often succinctly defined as data about data, metadata includes the descriptions that define the organization of data and information so that the interpretation of such data and information can be undertaken automatically by computer software. Metadata typically includes specifications for nomenclature (i.e., vocabulary), the structure of data in logical data models, taxonomies, interfaces, mapping tables, object models (i.e., ontologies), and business rules. The DoD Net-Centric Data Strategy (Stenbit 2003, 6-8) describes three principal mechanisms for storing and processing metadata, as follows. *Metadata Registries* are used to describe the structure, format, and definition of data. They may be implemented as a software application that uses a database to facilitate the storing and searching for data, definitions of data, relationships among data, and document formats. In this respect a Metadata Registry is like a library document that defines the kind of information that is required to be printed on the cards of a library card catalog (i.e., it does not describe any particular library holding, but only the kind of information that needs to be provided for all holdings). A *Metadata Catalog*, on the other hand, provides the information stipulated by the Metadata Registry for each individual set of data. This information is also typically stored in a database. *Shared Space* refers to the storage of the actual data that are described in the Metadata Catalog.

In the three-layer architecture of a typical knowledge management system shown in Figure 3.6, the Metadata Registries and Catalogs would reside in the Mediation Layer while the actual data (i.e., Shared Spaces) would be found in the Data Layer. The range of metadata that is planned for the DoD Metadata Registry includes: standard reference tables; XML, EDI, X-12, EBXML, and similar formats; logical data model structures; symbologies; taxonomies; ontologies; and transformation services.

MIP: Multilateral Interoperability Programme

NASA: National Aeronautics and Space Administration

NIST: National Institute for Standards and Technology

- NRC:** National Research Council
- OASIS:** Organization for the Advancement of Structured Information
- OLAP:** On Line Analytical Processing (OLAP) tools are applications that are typically utilized to extract answers to Who?, What?, and Why? queries, constrained by the need for users to understand the structure of the database since the contents of the database are not represented in terms of the user's domain.
- Ontology:** The term ontology is loosely used to describe an information structure, rich in relationships, that provides a virtual representation of some real world environment (e.g., the context of a problem situation such as the management of a transport corridor, the loading of a cargo ship, the coordination of a military theater, the design of a building, and so on). The elements of an ontology include objects and their characteristics, different kinds of relationships among objects, and the concept of inheritance. Ontologies are also commonly referred to as *object models*. However, strictly speaking the term ontology has a much broader definition. It actually refers to the entire knowledge in a particular field. In this sense an ontology would include both an object model and the software agents that are capable of reasoning about information within the context provided by the object model (since the agents utilize business rules, which constitute some of the knowledge within a particular domain).
- OSD:** Office of the Secretary of Defense
- RF:** Radio Frequency
- SLPC:** Single Load Planning Capability
- SMF:** Services Management Framework
- SOA:** Service-Oriented Architecture
- SOAP:** Simple Object Access Protocol
- UAS:** Unmanned Aircraft System
- UDDI:** Universal Description, Discovery and Integration
- UAV:** Unmanned Airborne Vehicle
- UGV:** Unmanned Ground Vehicle
- UMS:** Unmanned Maritime System
- USTRANSCOM:** United States Transportation Command
- UUV:** Unmanned Undersea Vehicle
- W3C:** World Wide Web Consortium
- WSDL:** Web Service Description Language
- XML:** Extensible Markup Language

XSLT: Extensible Stylesheet Language Transformation

10. References and Bibliography

- Biermann A. and J. Feldman (1972); 'A Survey of Results in Grammatical Inference'; Academic Press, New York.
- Blum A. and M. Furst (1997); 'Fast Planning Through Planning Graph Analysis'; Artificial Intelligence, 90, (pp. 281-300).
- Brown A., S. Johnston and K. Kelly (2003); 'Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications'; April 17, <<http://www.rational.com/media/whitepapers/TP032.pdf>> (30 May 2003).
- Cagle K., M. Corning, J. Diamond, T. Duynstee, O. Gudmundsson, M. Mason, J. Pinnock, P. Spencer, J. Tang, A. Watt, J. Jirat, P. Tchistopolskii, and J. Tennison (2001); 'Professional XSL'; Wrox Press Ltd., Birmingham, England.
- CDM (2004); 'Functional and Technical Basis of TRANSWAY: An Intelligent Toolset for Deployment and Distribution Operations Centers'; Technical Report, USTRANSCOM (J6-A), Contract DAMT01-02-D-0024, Delivery Order 0014, October (pp. 31-4).
- Cohen B. L. and C. A. Sammut (1978); 'Pattern Recognition and Learning With a Structural Description Language'; Proceedings Fourth International Joint Conference on Artificial Intelligence, IJCP, Kyoto, Japan (pp.394).
- 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.
- Diaz C., B. Waiters, J. Pickard, J. Naylor, S. Gollery, P. McGraw, M. Huffman, J. Fanshier, M. Parrott, S. O'Driscoll-Packer, B. Pendergrast and E. Sylvester (2006); 'ICODES: Technical and Operational Description'; Technical Report CDM-20-06, CDM Technologies, Inc., San Luis Obispo, California, November.
- DoD (2001); 'Department of Defense Dictionary of Military and Associated Terms'; Joint Publication 1-02, 12 April (as amended through 26 August 2008), Joint Chiefs of Staff, US Department of Defense, Washington, DC.
- Durfee E. (1988); 'Coordination of Distributed Problem Solvers'; Kluwer Academic, Boston, Massachusetts.
- Erl T. (2005); 'Service-Oriented Architecture (SOA): Concepts, Technology, and Design'; Prentice Hall Service-Oriented Computing Series, Prentice Hall, Englewood, New Jersey.
- Erwin S. (2008); 'Chokepoints: More eyes in the sky may not generate better intelligence'; National Defense, 1 June.
- Ewalt D. (2002); 'The Next Web'; Information Week, October, (www.informationweek.com/story/IWK20021010S0016).
- Freund Y. and R. Schapire (1997); 'A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting'; Journal of Computer and System Sciences, 55(1) (pp. 119-139).
- Friedman-Hill, E. (2003); 'JESS In Action'; Manning Publications Co., Greenwich, Connecticut.
- Ganek A. and T. Corbi (2003); 'The Dawning of the Autonomic Computing Era'; IBM Systems Journal, 42(1) (pp.5-18).

- GAO (2008); 'Unmanned Aircraft Systems: Federal Actions Needed to Ensure Safety and Expand Their Potential Uses within the National Airspace System'; United States Government Accountability Office (GAO), GAO-08-511, Washington, DC, May.
- Gil Y. and V. Ratnakar (2002); 'Markup Languages: Comparison and Examples, Information Sciences Institute, University of Southern California, TRELIS Project, (www.isi.edu/expect/web/semanticweb/comparison.html).
- 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.
- Grabner H. and H. Bischof (2006); 'On-line Boosting and Vision'; Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Volume 1 (pp. 260-7).
- Hinton G. (2007); 'To Recognize Shapes First Learn to Generate Images'; in Computational Neuroscience: Theoretical Insights into Brain Functions, Elsevier, New York, New York.
- Hinton G. and R. Salakhutdinov (2006); 'Reducing the Dimensionality of Data with Neural Networks'; Science 313, 5786, July (pp. 504-7).
- Hinton G., S. Osindero and Y. The (2006); 'A Fast Learning Algorithm for Deep Belief Nets'; Neural Comp. 18(7), July (pp. 1527-1554).
- Hinton G., C. Williams and M. Revow (1992); 'Combining Two Methods of Recognizing Hand-Printed Digits'; in Aleksander I and J. Taylor (eds.), Artificial Neural Networks II: Proceedings of ICANN-92, Elsevier North-Holland, Amsterdam, The Netherlands.
- Huang H. (2006); 'The Autonomy Levels for Unmanned Systems (ALFUS) Framework: Interim Results'; NIST Special Publication 1011, NIST, Washington, DC.
- Huang H., K. Pavek, B. Novak, J. Albus and E. Messina (2005); 'A Framework for Autonomy Levels for Unmanned Systems (ALFUS)'; Proceedings AUVSI Unmanned Systems North America, Baltimore, MD, June.
- Karsai G. (2000); 'Design Tool Integration: An Exercise in Semantic Interoperability'; Proceedings of the IEEE Engineering of Computer Based Systems, March, Edinburgh, England.
- Kennedy J. and R. C. Eberhart (2001); 'Swarm Intelligence'; Morgan Kaufmann, San Francisco, California.
- Koza J. (1992); 'Genetic Programming'; MIT Press, Cambridge, Massachusetts.
- Leighton R. and J. Undesser (2008); 'An Ontology Based Information Exchange Management System Enabling Secure Coalition Interoperability'; AFCEA-GMU Conference, Critical Issues in C4I, Fairfax, Virginia, 20-21 May.
- Lesser V. (ed.) (1995); 'Proceedings First International Conference on Multi-Agent Systems'; ICMAS-95, AAAI Press/MIT Press, Cambridge, Massachusetts.
- Levi K. and Y. Weiss (2004); 'Learning Object Detection From a Small Number of Examples: The Importance of Good Features'; Proceedings of Computer Vision and Pattern Recognition (pp. 53-60).
- McClelland J. L. and D. E. Rumelhart (1988); 'Explorations in Parallel Distributed Processing: A Handbook of Models, Programs and Exercises'; MIT Press, Cambridge, Massachusetts.

McGovern J., et. al.(2003); 'Service-Oriented Architecture'; in Java Web Services Architecture, April, <<http://www.developer.com/design/article.php/2207371>> (30 May 2003)

Minsky M. and S. Papert (1969); 'Perceptrons'; MIT Press, Cambridge, Massachusetts.

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

Nibecker J., D. Taylor, R. Chambers, H. Larsen, K. Cudworth, C. Warren, M. Porczak, and J. Pohl (2006); 'TRANSWAY: Planning with the Tabu Search Algorithm'; Focus Symposium on Advances in Intelligent Software Systems, Baden-Baden, Germany, August 8 (Pre-Conference Proceedings, June 2006).

NRC (2002); 'Future R&D Environments: A Report for the National Institute of Standards and Technology'; National Academic Press, National Academy of Sciences, Washington, DC.

Ojala T., M. Pietikäinen and T. Mäenpää (2002); 'Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns'; IEEE Transactions on Pattern Analysis and Machine Intelligence.

OSD (2007); 'Unmanned Systems Roadmap 2007 - 2032'; Office of the Secretary of Defense (OSD), United States Department of Defense (DoD), Washington, DC, 10 December.

OSD (2005); 'Unmanned Systems Roadmap 2005 - 2030'; Office of the Secretary of Defense (OSD), United States Department of Defense (DoD), Washington, DC, 4 August.

Patterson D., A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiziman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman and N. Treuhaff (2002); 'Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies'; UC Berkeley, Computer Science Technical Report (UCB//CSD-02-1175), University of California, Berkeley, California, March 15, 2002.

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

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

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., M. Porczak, K.J. Pohl, R. Leighton, H. Assal, A. Davis, L. Vempati and A. Wood, and T. McVittie, and K. Houshmand (2001); 'IMMACCS: A Multi-Agent Decision-Support System';

- Technical Report, CADRU-14-01, Collaborative Agent Design (CAD) Research Center, Cal Poly, San Luis Obispo, CA, June. (2nd Edition)
- Pohl J. (2005); 'Intelligent Software Systems in Historical Context'; in Jain L. and G. Wren (eds.); 'Decision Support Systems in Agent-Based Intelligent Environments'; Knowledge-Based Intelligent Engineering Systems Series, Advanced Knowledge International (AKI), Sydney, Australia.
- 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 Collaborative Agent Design Research Center (CADRC), Cal Poly (San Luis Obispo, CA) in Quantico, Virginia, June 5-7.
- Post E. (1943); 'Formal Reductions of the General Combinatorial Problem'; American Journal of Mathematics, 65 (pp. 197-215).
- Rosenblatt F. (1958); 'The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain'; Psychological Review, 65(6), Cornell Aeronautical Laboratory (pp. 386-408).
- RTCA (2007); 'Guidance Material and Considerations for Unmanned Aircraft Systems'; Radio Technical Commission for Aeronautics (RTCA), Special Committee 203, Washington, DC, 22 March.
- Sen S., M. Sekaran, and J. Hale (1994); 'Learning to Coordinate without Sharing Information'; in National Conference on Artificial Intelligence, (pp. 426-431).
- Siddiqui B. (2004); 'Web Services Security'; March,
<<http://webservices.xml.com/pub/a/ws/2003/03/04/security.html>> (5 Oct 2004)
- SPAWAR (2002); 'IMMACCS: Limited Technical Assessment (LTA)'; SPAWAR Systems Center, San Diego, California, 24-26 September.
- Stenbit J. (2003); 'Department of Defense (DoD) Net-Centric Data Strategy'; US Department of Defense, Chief Information Officer (CIO), Washington, DC, May 9, 2003.
- Veloso M., J. Carbonell, A. Perez, D. Borrajo, E. Fink and J. Blythe (1995); 'Integrating Planning and Learning: The PRODIGY Architecture'; Journal of Theoretical and Experimental Artificial Intelligence, 7(1).
- Viola P. and M. Jones (2001); 'Rapid Object Detection Using a Boosted Cascade of Simple Features'; IEEE Computer Society Conference on Computer Vision and Pattern Recognition, December.
- Weiger R. (2007); 'Military Unmanned Aircraft Systems in Support of Homeland Security'; USAWC Strategic Research Project, US Army War College, Carlisle Barracks, PA, 30 March.
- Winston P. H. (1970); 'Learning Structural Descriptions from Examples'; Technical Report AI-TR-231, MIT, Cambridge, Massachusetts, September.
- 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. and Collaborative Agent Design Research Center, Cal Poly, San Luis Obispo, California, December.

Wooldridge M. and N. Jennings (1995); 'Intelligent Agents: Theory and Practice'; The Knowledge Engineering Review, 10(2) (pp.115-152).

Yee W. and G. Hinton (2000); 'Rate-Coded Restricted Boltzman Machines for Face Recognition'; Gatsby Computational Neuroscience Unit, Proceedings NIPS-00.

Zang M., A. Gray and M. Hobbs (2008); 'Similarity Assessment Techniques'; Pre-Conference Proceedings, Focus Symposium on Intelligent Software Tools and Services, InterSymp-2008, Baden-Baden, Germany, 24-30 July.

Zang M. and C. Neff (2003); 'SILS MRAT: A Shipboard Integration and Mission Readiness Analysis Tool'; 13th Ship Control Systems Symposium (SCSS), Orlando, Florida, April 7-9.