(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
7 August 2003 (07.08.2003)

PCT

(10) International Publication Number
WO 03/065173 A2

(51) International Patent Classification[7]: G06F

(21) International Application Number: PCT/US03/03085

(22) International Filing Date: 3 February 2003 (03.02.2003)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/353,487 1 February 2002 (01.02.2002) US

(71) Applicant and
(72) Inventor: **FAIRWEATHER, John** [US/US]; 1649 Wellesley Drive, Santa Monica, CA 90405 (US).

(74) Agents: **THIESSEN, Kendall, I.,** et al.; Gibson, Dunn & Crutcher LLP, 1801 California Street, Suite 4100, Denver, CO 80202 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Declaration under Rule 4.17:
— *of inventorship (Rule 4.17(iv)) for US only*

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: A SYSTEM AND METHOD FOR MANAGING KNOWLEDGE

(57) Abstract: An intelligence system is provided that is comprised of the following basic components. First, a system for converting incoming unstructured data into a well described normalized form. Since the incoming data is multimedia and may represent some data type for which support is provided by the underlying OS platform, this normalized form includes the ability to fully describe and manipulate arbitrarily complex native or non-native binary structures and collections. This support is preferably provided by a dedicated 'mining' language tied intimately to a system ontology. Second, a system for accessing and manipulating data held either in memory or in persistent storage in its normalized binary form so that small executables, or 'widgets', within the system can freely and effectively operate on data types they have never before encountered simply by knowledge of the 'type' of data involved. Third, an 'ontology' or world model that represents and contains the items and fields necessary for the target system to perform its function. The ontology would preferably fully specify the form of the normalized binary data. Fourth, a memory system, tied to the ontology, which defines the structure of and access to any persistent storage containers that are required to contain the data. Fifth, a memory management system for splitting incoming data into those portions to be directed to each container. Sixth, a query system for querying each container to retrieve portions of such a composite object. Preferably, all database tables and queries are auto-generated from the ontology, thereby eliminating the role of the conventional Database Administrator (DBA). Seventh, a Ul to display and interact with data within the system. In the preferred embodiment, the UI is automatically generated and its behaviors automatically handled by the underlying substrate thus removing this programming burden from the developer (thereby largely eliminating the role of the GUI programmer). Finally, a memory system that forms collections of datums, and enables manipulation and exchange of these collections both within the local machine as well as across the network. In the preferred embodiment, such collections support the ability to attach arbitrary tags or annotations to the binary data they contain without in any way altering the binary representation itself. Additionally, the system supports the concept of either null or dirty (i.e., has been changed locally) datum.

WO 03/065173 A2

A SYSTEM AND METHOD FOR MANAGING KNOWLEDGE

Inventor: John Fairweather

BACKGROUND OF THE INVENTION

Historically, a major problem with designing complex knowledge representation systems has been the difficulty of acquiring the necessary data in a structured form that algorithms representing the specific 'application' can process, and thus produce useful results. The traditional solution has been to restrict such systems to applications where the data is available within a database, normally relational and accessed using Structure Query Language (SQL). By applying these restrictions, the system design problem becomes tractable, and many useful but limited and localized calculations can be performed.

In the overwhelming majority of cases, data gets into such a database by manual data entry. This requires a highly structured environment where an operator is led through the process of entering all the necessary fields of the database 'tables' by a user interface (UI) component that has been tailored to the particular application, and which thus embodies the know-how necessary to ensure correct data entry.

In recent years, however, technologies such as B2B suites and XML have emerged to try to facilitate the exchange of information between disparate knowledge representation systems by use of common tags that may be used by the receiving end to identify the content of specific fields. If the receiving system does not understand the tag involved, the corresponding data may be discarded. These systems simply address the problem of converting from one 'normalized' representation to another, (i.e., how do I get it from my relational database into yours?) by use of a tagged, textual, intermediate form (e.g. XML). Such text-based approaches, while they work well for simple data objects, have major shortcomings when it comes to the interchange of complex multimedia and non-flat binary data. At a minimum, an interchange language designed to describe and manipulate binary data must be implemented, but current approaches fail to take this crucial step. Systems that operate in a domain where the source and destination have explicit or implicit knowledge of each other, or in which endpoints, to facilitate and enable interchange, comply with a standardized exchange format, we shall call 'Constrained Systems' (CS). The vast majority of systems in existence today are constrained systems. Despite the 'buzz' associated with the latest data-interchange techniques, such systems and approaches are totally inadequate for addressing the kinds of problems faced by a system, such as an intelligence system, which

attempt to monitor and capture streams of unstructured or semi-structured inputs, from the outside world and derive knowledge, computability, and understanding from them.

Once the purpose of a system is broadened to acquisition of unstructured, non-tagged, time-variant, multimedia information (much of which is designed specifically to prevent easy capture and normalization by non-recipient systems), a totally different approach is required. In this arena, many entrenched notions of information science and database methodology must be discarded to permit the problem to be addressed. We shall call systems that attempt to address this level of problem, 'Unconstrained Systems' (UCS). An unconstrained system is one in which the source(s) of data have no explicit or implicit knowledge of, or interest in, facilitating the capture and subsequent processing of that data by the system.

Nowadays, the issue faced by any unconstrained system is not the lack of data but rather the flood of it. Digital information, mountains of it, is available everywhere. It floods the Internet (whose information contents by some estimates doubles every few months now), it fills the airwaves as phone calls, radio and video transmissions, e-mails, faxes, dedicated data feeds, databases, data streams, chat rooms, corporate networks, banking systems, peer-to-peer networks, bulletin boards, web pages, stock markets, telexes, etc. The problem now is that no system can handle the torrent of data that flows through the digital world we have created. The best that can be achieved is to sample some of the current as it washes by, and look for items of interest or significance within it. Even a small sample of such a stream represents a torrent that would overwhelm a conventional constrained system within seconds.

The basic configuration of an intelligence system is that digital data of diverse types flows through the intake pipe and some small quantity is extracted, normalized, and transferred into the system environment and persistent storage. Once in the environment, the data is available for analysis and intelligence purposes. Any intercepted data that is not sampled as it passes the environment intake port, is lost.

The information to be monitored is not just simple text, it is multimedia sounds, images, videos, compound documents etc. It is unstructured. It is multilingual. Most of what occurs in the world, does not do so in English. Information quality varies widely. Much of what is transmitted is garbage, wrong, or simply represents rumor or uninformed opinion. Knowledge of the source of the information must dictate its interpretation. The conventional assumption that the value of a field is exact and can be stored in a single box or cell simply

does not apply. Even if the captured data can be regarded as absolute, its interpretation is a matter of opinion among those analysts using the system, and thus its value can be modified depending on the domain or perspective of the user of the data.

Most of the information available on the web is low-grade, unreliable information placed there to further somebody's agenda, not to provide truth. Indeed, most 'reliable' or high grade open-source information comes from publishers of one sort of another, and these people have little or no incentive to place such information on the web given the lack of any workable business model for making money from information so posted. As a result, worthwhile information must be intercepted, or for open-source data 'mined,' from a multitude of other sources, many designed to make such extraction more difficult in order to preserve the publisher's intellectual property. Thus, Lexis/Nexus for example has thousands of high grade databases totaling more than 25 times the total data content of the web at this point, which can be accessed and searched (in a limited manner) only via a subscription account. News and reporting services all have different delivery formats, equipment, and media. An intelligence system must accommodate this diversity of sources as well as providing for custom, intercepted, and private feeds available only to a specific organization. Crawling the web, while enlightening, and certainly an important capability, is not a complete answer to intelligence, to in-depth research and analysis, or to the extraction of meaning. A datum coming from a given source must maintain a reference to that source since this will later determine the reliability placed on that datum should it contribute in any way to an analytical conclusion.

To further complicate the issue of data sources, in intelligence applications, the identity and reliability of the persons involved in an intercept is frequently unknown or questionable. Additionally, the true identity and nature of entities referred to via key phrases or aliases in the intercept may be unknown, and may indeed be the subject of the analyst's investigation. Even known entities are frequently referred to via aliases. Thus, to perform analysis the system must support the concept of partially resolved references to data. That is, aliases to entities or things that have not yet been assigned to a known datum in the system. Thus, if the participants in an exchange refer to the 'client,' it becomes important to establish who that client is. However, since the word 'client' may appear in a myriad of different contexts where it actually refers to completely different entities, we must extend the concept of a source to incorporate the concept of a 'source domain' identified either by the persons

involved in the intercept, or by other means. Within this 'domain' the word 'client' is assumed to correspond to a given entity, possibly still unresolved. Outside this domain the word will have other connotations. The underlying architectural substrate must provide for and support this type of ambiguity

In a UCS, information is transitory. Once it has been transmitted, intercepted, and has flowed through the pipe, it is gone. It cannot be retrieved later from a web page or database engine. Because the information is transitory, it is essential that any monitoring system be able to identify it as important as it passes through the system intake pipe so that it can be selectively captured from the stream for subsequent analysis. Due to the huge volumes involved, not all data can be stored persistently and so reliable and automated sampling of the passing stream is a prerequisite. Moreover, the answer to any given question varies with time, and spotting these variations and the patterns they represent is the essence of intelligence. Again a conventional database is ill-suited to the demands of such time-variant data.

Rich multimedia data is full of subtleties, contextual overtones, and fine detail that cannot be captured as 'fields,' thus it is essential that data captured for storage and analysis be preserved in its entirety. The integrity of the original data must not be compromised by the conventional process of shredding it into standardized relational fields. To do so may remove the most important ingredient of the data. On the other hand, without some kind of field-like partitioning, no useful computation can be done, so a system must do both. That is, the data may be stored multiple times in different forms and containers. Furthermore, in multimedia data, each aspect of the data is best suited to analysis, search, storage, and distribution by different 'containers.' For example large bodies of text are best handled and searched by inverted file type text engines whereas fixed numeric or descriptive fields rightly belong in a relational database. Image, video, maps, sounds, and other multimedia fields must be stored, distributed and searched using engines, processes, and hardware that are best suited to the needs of the particular type, and thus the system must support a variety of 'containers' targeted at different media types and processes. A fingerprint or face recognizer capability obviously belongs in a different container than relational fields relating to specific fingerprints or images. To attempt to force all such tools into the framework of a common container, presumably a relational database, would be cost-prohibitive and extraordinarily inefficient.

Having taken the step of dispersing aspects of a given data item to the various containers that most effectively deal with those aspects, it becomes obvious that the system must now have the ability to seamlessly and transparently re-assemble those aspects back into the appearance of a unified whole for presentation to the user. Furthermore, the system must now provide a unified framework for querying the various aspects according to the querying concepts that make sense for the aspect involved, reassembling the results of various aspect specific portions of a query into a unified hit-list of results. Thus, for example, a fingerprint query would be specified and then routed to an entirely different container and engine than would other aspects of the same query such as the time period involved, or the physical region within which the search is to be constrained. These latter two aspects should be routed to relational and geographic container/query engines respectively. The need for a unified and extensible, distributed query language becomes readily apparent, as does the need for an auto-generated UI environment capable of smoothly stitching together the various components of whatever data is finally retrieved.

The nature of the intelligence problem is that most of the time you do not know what you are looking for until you find it, often much later. However, when you have identified the significant aspect, it suddenly becomes necessary to do a detailed analysis of all past data to examine the newly significant aspect to see if there are similarities or trends. Thus, the 'data-model' for the system is subject to continuous change on an analyst-by-analyst basis as they pursue divergent lines of inquiry into finding the key to some event of interest. What is needed, then, is a system designed for intelligence purposes that accommodates this behavior. Again, conventional systems fail to address this dynamic data-model issue.

Supposing one could automate the capture of large quantities of the digital world's data stream and deliver it to many analysts whose task was to search the stream for significance and meaning; still the volume of data would overwhelm all but the largest installations. This is because human beings have evolved sensors and mental apparatus to deal with the unique characteristics of information as it is presented to us in the analog world in which we live. In this world, the relevance of information generally falls off exponentially with distance from the observer (both in space and time), and as a consequence all of our senses exhibit a similar falloff. We take advantage of this fact to limit the amount of data we need to process. Furthermore, the same is true of our minds; that is, we are able to apply 'logical thought' only to the one thing that is our current focus. Our senses compete to filter

everything we observe (based for the most part on distance or apparent magnitude) so that the most important item is brought to our attention at any given time for processing. When asked to give a description of what has happened to us in the last few minutes, each observer will give a different answer, and that answer actually corresponds to a listing of the mental models that were triggered by the focus, and the order in which they occurred. This frequently yields a very different history to what occurred in actual reality, and accounts for the notorious unreliability of most witnesses.

Unfortunately, in the digital domain, there is no exponential relevance decay phenomenon. Events occurring anywhere in the world may be as relevant to us as those occurring nearby. The analyst is forced to consider anything that may be potentially relevant regardless of spatial, temporal, or conceptual proximity. The result, given the volume of data, is information overload. Moreover, digital information environments such as the web are designed to capture and lead the focus of the person using them, primarily to garner advertising dollars. Thus, we have all experienced the problem of searching for the answer to something on the web, only to be forced into the focus of the web sites we look at, with the result that eventually, hours later we give up, having failed to find what we were looking for, or more likely, having forgotten entirely what it was in the first place. Again, this effect occurs because the digital domain is not constrained by the same falloff law that our analog world is. Each navigation step may be arbitrarily large, and our minds are poorly equipped to maintain focus, and thus search for meaning or relevance in this environment. Thus, a primary goal of any UCS must be to help the analyst maintain focus and empower him to direct his inquiries based on his analytical goals (see Patent ref. 8). To do this, the system must gather and pre-filter information to present only the most relevant portions while accentuating and visualizing the relationships between adjacent data (spatially, temporally, or conceptually) so that the sensors and mental models we all use can be applied to best advantage to analyze that data for patterns, trends, or anomalies. Such pre-filtering must be completely tailored on a per-analyst basis since the filters must be digital representations of the mental models that particular analyst has built up in order to categorize and thus process events.

In effect, such a UCS must enable the analyst to construct or specify, over time, a digital alter ego which he empowers to be his representative in the torrent of information passing through such a system, and which is authorized to some level to filter and pre-process

information, thus leaving the analyst free to make the non-linear leaps and connections that so uniquely characterize human thought. Many attempts have been made in the past to create such avatars, bots, or intelligent agents, mostly by the application of artificial intelligence techniques to specify a rule base that represents, in some way, the thought process of the analyst. Except in restricted domains, all such attempts have largely failed because human thought is not simply the repetitive application of a rule set. Indeed, we still have little idea how to model what we do when we solve a problem, and certainly the techniques we use are unique to each individual and more a result of experience, prejudices and judgment than they are the application of internal rule sets. This inevitably leads us to the conclusion that an architecture for a UCS must through some easy, presumably graphical means, allow each analyst to specify his personal analytical techniques out of whatever building blocks from whatever technical domain or technique he deems relevant. Some kind of visual wiring language where the information passing through the connecting flows represents data gleaned from the captured flow, and the blocks being connected represent limited and specialized processing blocks, is required. Once so specified, an analytical technique must be able to be launched on an automated basis into the intake stream in order to look for matching data to be brought to the attention of interested analysts.

Central to the ability to analyze new information as it passes by us, is the fact that we are essentially the sum of our experiences. It is our ability to build mental models that allow categorization and processing of new information that constitutes what we call intelligence. A critical aspect of this ability is the need for a large and related experience base that can be used to mentally model and predict the outcome of potential actions in order to choose between alternatives. In the digital domain, if we are to analyze a deluge of data, the same is true, that is, only by building up a vast and encompassing history of past events and their consequences can we begin to understand the potential relevance and consequences of new events appearing in the intake pipe. For even a moderately sized UCS, this represents a storage requirement in the Terra-byte or Peta-byte range given the multimedia nature of the inputs. More important however is the fact that due to the diverse nature of the feeds, and because in any practical system for monitoring global events, feeds must be acquired globally, at the source. It becomes apparent that this storage must be distributed, and must be closely tied to the architecture of the acquisition intake. This acquisition server architecture must, of necessity, be distributed given the physical separation of feeds. Further, given the demanding storage and isochronous retrieval requirements of rich media types such as video,

it is apparent that deep storage architecture and access must be tailored to exactly match such a distributed server architecture on a per data-type and per-feed basis.

The concept of using the sum of our experiences as a kind of lens with which we view the world is key to understanding why systems claiming to provide such buzzword capabilities as "Asset Management" or "Knowledge Management" are only peripherally related to the intelligence problem itself. An asset or knowledge management (KM) system is engaged in the process of looking inwards into an organization to understand and control what is within. An intelligence system does this also, but then uses the knowledge gained by this experience and examination as a lens to allow interpretation of new information coming from the outside world. In effect, we use what we know and learn about ourselves to help us interpret what we see. In the KM case, the data pool is largely static, structured, and controllable. In the intelligence system case, the pool is simply an eddy in a rushing torrent where control of the torrent is out of the question. KM systems are in reality nothing more than thin veneers over relational databases, an approach that is wholly inadequate to the needs of an unconstrained intelligence architecture.

The purpose of an intelligence system is to facilitate the analysis of captured data and allow the rapid and effective distribution of such analyses to the intelligence consumers (i.e., 'clients') of such a system. Once the system involves multimedia information, the conventional solution of printing out a paper report and hand delivering it to the client becomes wholly inadequate. Multimedia information cannot be well represented on paper, and yet as the saying goes, a picture is worth a thousand words. What then is a video segment or sound recording worth? The truth of the matter is that multimedia data types are able to convey a much richer and more impactful presentation than words alone can. Thus, it is incumbent on such a system to design in the ability to easily create and electronically deliver full multimedia reports to its clients. This means that the report must actually be a working 'application' capable of full interaction with the client, and when necessary retrieval and playback of any multimedia and other components from archival storage within the system. Creation of such reports must be a relatively trivial matter for the analyst(s) involved. Delivery of multimedia reports without the ability for those reports to access data from system storage would not be nearly as effective. Furthermore, by taking this approach, one opens the door to regarding the report as a custom portal for the information consumer client to examine the details of a particular issue, review the backup data that lead to the

reports conclusions, and to draw additional conclusions regarding, or obtain additional details relating to, the subject matter as necessary. Thus, an intelligence architecture should be designed to be end-to-end; that is, it must handle every stage of the process from capture, storage, indexing, search, analysis and finally to presentation. Often decision makers or information consumers are unskilled in the use of computers, and so a simpler (possibly hands-off) kiosk or web-portal like end-user mode, in addition to the more extensive normal analytical mode, must be provided. This mode must anticipate the needs for projection on large screens and the likelihood that multiple individuals will be in the audience. Access security, possibly using biometrics is an issue.

In adopting an architectural, rather than an application driven approach to solving the problem of unconstrained systems, a prerequisite is that the architecture provide a complete suite of tools to allow the end user to customize and extend the system by adding new tools and analyses as desired. Any approach to implementing a UCS that is not predicated on allowing the system staff to extend and modify the environment in arbitrary ways will not only be forced to severely constrain what is possible, but will also be so complex to define and subsequently implement that it may never work. Therefore, given that such customization is not only allowed, but encouraged, it is quickly apparent that a matching set of debugging tools must also be provided in order to make such customization practical. The system itself must expose a large and complete Applications Programming Interface (API) to allow development at the low level. Development however, must be possible on at least two levels. For the purposes of software engineers, whose goal is to integrate new capabilities seamlessly into the existing environment, code level support and APIs with detailed documentation is required. As much as possible of the detailed and housekeeping work must be handled automatically within the environment so that code level programmers can focus purely on the algorithm they wish to implement, not on such things as UI, communications, data access etc. For the purposes of analysts, who generally are not programmers, but who nonetheless need to express and specify analytical processes in terms of data flowing between a set of computational blocks, a visual programming language must be provided.

The issue of multilingual data is also a key hurdle to be overcome in any practical intelligence and monitoring system. The reality is that most interesting 'events' first appear in some local, probably non-English source and only later after capture and refinement by others does the information appear in English from another secondary, tertiary, or more

indirect source. At each step of this process, 'integrity' and nuances of the original source are degraded and lost. Any practical system must thus be capable of capture at the source and in the language/format of the original. Mechanisms must be developed to handle and process the information in a productive and speedy manner despite the fact that the associated text may not be in English. There may be no time for a full translation during the brief transit period of the data through the system intake pipe. Failure to address this issue would mean all data must be centralized for formal translation prior to processing, and this requirement would obviously clog the intakes of any installed system targeted at even a moderate sized multi-lingual stream.

Non-English languages pose many problems that are trivially addressed in English. Foremost among these problems is the issue of 'stemming' or finding the root word or meaning of a given word. In English, stemming to extract the root word is trivial. One simply chops off common trailing modifiers to obtain the root word. Thus, in an English language search "Teachers" and "Teaching" are both trivially and automatically stemmed to yield the root word "Teach" and it is this that is actually searched (at least in non-trivial text search engines). In other languages, for example Arabic, each word may represent a mini-sentence. Thus, in Arabic "he taught them" or "they taught us" might be represented by single but very distinct words. The root word is not immediately apparent by examining the actual characters since even the characters involved in such mini-sentences are different. Meaningful search in many non-English languages is thus a subject of research since the Roman script derived language concept of a "key word" has little meaning in many other scripts. A key problem that must be addressed by a practical intelligence architecture is therefore how to stem foreign language inputs to allow meaningful word associations and "concept" queries to be made, while still allowing exact match searches where necessary or appropriate. Failure to address this problem makes the system virtually useless for many foreign script systems.

Multilingual requirements impact not only intake processing, but more obviously the user interface to the system, which must have the inherent ability to translate dynamically and on the fly between languages and appearances depending on the language or wishes of a particular user. The process of modifying a software program to appear and behave correctly in another language or script system is known as 'localization,' and is a multi-billion dollar industry and a major headache for all developers of software who wish to target foreign

markets. Localization of a software product can take months, requires extensive source code changes or accommodations, and must be repeated (at vast expense) every time a new upgrade is released. One requirement of an unconstrained intelligence system is the ability reduce this localization process to an automatic and instantaneous behavior which is not in any way tied to the code that is generating or handling a particular aspect of the UI. If such a tie in did exist, the ability of the system to adapt globally (i.e., in a multilingual manner) to changes would be hampered by the rate at which localization could take place, and inevitably portions of the system would become inconsistent with other portions.

In any large collection of disparate data, the problem of how to navigate around it effectively becomes critical. We see that in the only successful example of a truly complex system, the Internet, the approach taken to navigation was to implement embedded hyperlinks which transition the users focus to the referenced URL. This works effectively, but is an incredibly manual, restrictive, and error prone business. The web-site designer must hand-insert the chosen hyperlink to the URL, thereby enforcing his perspective on the user rather than that of the user himself. Worse yet, URLs change continuously and the referencing link then becomes out of date and useless. What is needed in a UCS is the ability to define and enable/disable hyperlink domains on a per-user basis, and to have those hyperlinks automatically applied to every bit of textual data present in the system or displayed to the user. In other words, we need a dynamic hyperlinking architecture under the control of each user, not of the information source. This directly addresses the loss-of-focus issue discussed earlier by allowing the user to define and modify his own hyperlinking environment. The architecture and the UI it presents must provide and automate this facility. When a hyperlink is clicked, the architecture must be able to identify the nature and location of the datum to which that hyperlink refers, and to automatically launch the appropriate display behaviors to show the target datum to the user in the most appropriate manner.

Given a distributed UCS through which large quantities of data will be passing, not only as it is ingested, but also as it is passed between various analytical processes, it is apparent that efficient representation of that data and its relationships in binary form must be supported by the environment. Most data is not 'flat', that is it comprises many chunks of variable sized memory which refer to each other via pointer or similar references. As it becomes necessary to pass such data from one process or machine to another, the data must be 'flattened' into a single contiguous chunk for transmission and then 'unflattened' at the

other end into its original form. This process is known as serialization (and de-serialization). All present data interchange environments are forced to perform serialization and de-serialization every time data is exchanged between processes. As the amount of data involved increases, the processing overhead of the serialization/de-serialization cycle begins to dominate until one reaches a practical limit in the amount of data that can be exchanged and the rate of such exchange. Unfortunately, with present day machines this limit is far below what is required for even a moderate UCS. Any architecture for unconstrained systems must therefore find a way to eliminate the serialization problem in its entirety.

The basic questions that are asked of an intelligence system can be summarized as "who", "what", "why", "when", and "where". The answers to most of these questions cannot be expressed as a column of numbers or text since the answer itself may not be in the data but must instead be deduced or visualized by the analyst. An unconstrained environment must support the pervasive use of a large and ever expanding set of visualization tools. Certain visualizers should clearly be built into the environment and have commonly accepted appearances. The visualizer to answer the question "where" for example is generally a map and associated Geographic Information System (GIS). The environment must provide such a GIS built-in. Going back to basics, the standard visualizer for displaying the results of a database query is the list, though we may not normally think of this as a visualizer. The environment must provide a basic list capability including the ability to display arbitrary, possibly media rich columns, and to sort on those columns. The basic list must be capable of handling data organized in arbitrary hierarchies. Other environment (or underlying OS) supplied visualizers must exist for the common rich media types (i.e., images, sounds, and video). Complex graph and chart plotting is of course a basic visualization capability and must be built into the environment. The ability to define arbitrary exotic visualizers to aid in detecting patterns, trends, and anomalies must be supported. Since many such visualizers (including any truly useful GIS visualizer), require a 3-D world to express as many connections and nuances as possible, we are lead to the conclusion that the UI environment for the architecture should be based on (or support) a 3-D standard. Given the fact that gaming demands are pushing computer equipment manufacturers to incorporate faster and faster 3-D graphics chips, we must conclude that the UCS UI environment would preferably be based on a 3-D software standard such as OpenGL that, like gaming engines, can take advantage of this hardware.

Focusing for a moment on the needs of a generalized GIS visualizer, consistent with our general UCS principals, it must permit the visualization of positional data in a variety of ways. Unfortunately, most, if not all, standard GIS systems suffer from a serious shortcoming in this regard. The problem is, that in order to be able to render maps in a reasonable time, GIS environments must eliminate the incredibly compute intensive process of performing the necessary projection calculations on every point in the map. These calculations involve 3-D transformations using transcendental functions that for a detailed large scale map are slow on present day commercial hardware. To overcome the problem, GIS systems pre-project their maps, and all map overlays, into a given projection (usually Mercator) so that the rendering of the maps to a client window does not involve the projection calculations. Unfortunately, there are large numbers of possible map projections and each of them has particular utility for visualizing different aspects of the information being projected. High end mapping systems may hold map data in multiple projections, but this requires storage many times that of the basic map data, and cannot in any case cover all possible projections or vantage points. This means for example that when one wishes to switch projections on the fly, or alternately to overlay data in one projection (a satellite image perhaps) on another (Mercator say), one is forced to go through a lengthy re-mapping process first. If multiple overlaid projections are involved the situation becomes untenable. The ideal UCS GIS system should find a way to store/render the data in its raw latitude/longitude format and do the projections on the fly.

In intelligence, the analyst needs the ability to visualize relationships between data, not only along well defined axes (e.g., space and time), but also along arbitrary axes defined by the analyst himself. Examples of such axes might be "Adverse actions towards the US", or "Activity relating to drugs". Clearly, the analyst must be provided with a way to define new arbitrary axes, and to specify through some arbitrary computational means, how one should determine the intercepts for a given datum on each of these axes. Once this information is known for a given collection of data, it is relatively easy to see how graphical visualization tools can be used to good effect to look for patterns, trends, and anomalies appearing along or between a particular set of such axes. The architecture must therefore support the ability to define such axes and rapidly determine coefficient vectors for any arbitrary set of data being visualized. Because such axis computation may be computationally expensive, doing it on the fly would drastically reduce visualizer responsiveness. For this reason, the architecture would preferably provide and support the

concept of a "vector server" responsible for continuously maintaining and updating coefficients for all data in persistent storage along whatever axes are currently defined. As data is fetched for visualization, the required coefficients can also be rapidly fetched from such a vector server by the visualizer. These coefficients would also form a key part of the solution to maintaining, examining, and acting upon non-explicit relationships between different system datums. It is important to understand that unlike conventional graphing axes, these arbitrary axes are non-orthogonal, each axis may be in some way related to many others. This fact can be taken advantage of to address the basic intelligence problem of not knowing exactly what one is looking for. If we imagine two related axes, one known (A) and one unknown (B), then as part of un-related work, an analyst may see the 'shadow' of a trend or anomaly related to B on the A axis, and may then be motivated to examine the causes behind this shadow, thereby discovering the existence and significance of the hitherto unexplored B axis. By subsequently defining a B axis to the system and then re-examining data in this light, new insights and relationships may become clear. This is a key aspect of the intelligence process that is not well supported by existing systems.

It is essential that the system user interface provided to the analyst take the form of a multimedia 'portal' which can be reconfigured and changed on a per-analyst basis using a simple graphical metaphor. Each analyst may in fact use multiple portals depending on the nature of the task at hand. This capability must be supported by the environment. Portals can be assembled out of any of the building blocks registered with, or provided by, the environment. The images presented above and in other patents referenced by this one combined with the technology revealed in patent ref. 11 make it clear how this portal capability can be implemented. The image below is of an 'executive mode' variant of the same basic portal illustrated elsewhere in order to show that UI appearance can be drastically varied without any impact on the underlying implementation or building-blocks.

Given the scale of the problem, it is clear that we are talking about a highly distributed architecture, even individual servers must clearly be implemented as distributed clusters. Equipment changes (and breaks), the environment changes, users move and change, as do the preferences of each user over time. It is clear then that the environment must provide extensive support for the re-configuration of any system parameter that might change. Such preferences span the range from the numbers and location of machines making up a given server cluster and the equipment to which they are connected, to the font a user

prefers or the color he likes to see buttons displayed in the UI. APIs and interfaces to access, distribute, and manipulate these preferences must also be provided. The goal of an environment should be to support dynamic and on-going reconfiguration of any target installation all the way from a single machine portable demo (if practical), to a worldwide distributed system and all its connected equipment, without the need to change a single line of compiled architectural code. Obviously, this goal is unattainable with most conventional approaches.

Having determined that we need an architecture that supports distributed server clusters, we should further ask ourselves what do we mean by a sever, and what is a client, in such a system. In conventional client/server architectures a server is essentially a huge repository for storing, searching, and retrieving data. Clients tend to be applications or veneers that access or supply server data in order to implement the required system functionality. In an unconstrained intelligence architecture, servers must sample from the torrent of data going though the (virtual) intake pipe. Thus it is clear that unlike the standard model, we will require our servers to automatically and in an unattended manner create and source new normalized data gleaned from the intake pipe and then examine that data to see if it may be of interest to one or more users. We need every server to have a built in client capable of sampling data in the pipe and instantiating it into the server and the rest of persistent storage as necessary. Thus we have little use for a standard 'server' but instead our minimum useful block is a server-client pair. As to the nature of the server portion itself, since each server will specialize in a different kind of multimedia data, and because the handling of each and every multimedia type cannot be defined beforehand, we see that we need a server architecture where the basic behaviors of a server (e.g., talking to a client, access to storage, etc.) are provided by the architecture but at any point where customization to server behaviors may be required, the server must call back to a plug-in API that allows system programmers to define these behaviors. Certain specialized servers will have to interface directly to legacy or specialized external systems and will have to utilize the capabilities of those external systems while still providing behaviors and an interface to the rest of the environment that hides this fact. An example of such an external system that must be masked behind our modified definition of a server might be a face, voice, or fingerprint recognition system. Thus the classic model of a big fat predefined server (a la Oracle etc.) that is purchased "as is" from a vendor, and wherein only the clients to that server can be changed by customer staff, does not apply to a UCS. Furthermore, at any time new servers

may be brought on line to the system and must be able to be found and used by the rest of the system as they appear. This requirement combined with our server-client building block starts to blur the line between what is a server and what is a client. Why shouldn't any 'client' machine be able to declare its intent to 'serve' data into the environment, indeed in a large community of analysts, over time this ability is essential if analysts are to be able to build on and reference the work of others. Thus every client must also potentially be a server. The only real distinction we can draw between a mostly-server and a mostly-client is that a server tends to source a lot more data on an on-going basis than does a client. An unconstrained network architecture must therefore be more like a peer-to-peer network than it is a classic client/server model. Application code running within the system should remain unaware of the existence of such things as a relational database or servers in general if such code is to be of any general utility. What we need then is some kind of automatic environment mediated and abstracted tie-in between the definition of the data within the system, and the need to route and access all or part of that data from a distributed set of servers.

Given the intense computational and processing requirements represented by a UCS, it is clear that we cannot afford the overhead or limitations of such cross-platform interpreted languages as Java. The system must therefore be based on one or more underlying OS platforms which are accessed from the environment via direct, efficient, compiled code. Since platforms may change, and differ from each other, the architecture must provide, wherever possible, a platform independent abstraction layer to which API level application programmers can write. The UCS architecture in effect becomes its own operating system (OS), layered on top of a conventional operating system and targeted specifically at providing OS type features related to the requirements of unconstrained systems. Since we must break computation up into large numbers of smaller, autonomous, computing blocks, which exchange data (and messages) through the substrate, it is clear that a highly threaded environment is required. This cannot be a monolithic deterministic application (see Patent ref. 11). Because we must pick a given OS architecture, the system should support the ability to deliver to, and interact with, its UI on a variety of client platforms perhaps via a less extensive UI set (such as a web page) or alternatively by interacting through a cross-platform GUI layer.

The analyst workload will of course require the use of a number of other commercial off-the-shelf (COTS) packages. Things like word processors, spreadsheets, Internet browsers, e-mail, sound and video editors, image analysis tools etc. The analyst needs all the same tools that a normal computer user does as well as, and in close conjunction with, the UCS environment. As a practical matter, it is clear then that the choice of platform on which to build an architecture is thus limited to the two consumer level OS platforms available, namely Windows and Macintosh. Any useful UCS architecture must be capable of treating COTS software applications as building blocks in the creation of processes within the system, we do not want to re-invent everything that is provided by all the COTS applications. Thus it must be possible in the architecture to 'wrap' a COTS application in a proxy process that exists within the environment so that the functionality that application provides can be utilized in an automated and scripted manner within the environment. Ease of such application scripting is a consideration in choosing the underlying OS. Given the multimedia nature of the information in an intelligence UCS, excellent and pervasive multimedia capability in the underlying OS platform is obviously crucial. Another consideration is the level and pervasiveness of that OS's (and its COTS applications) support for foreign languages and scripting systems. OS level security is another key factor. Finally, we must consider the range of COTS solutions available on the platform. In the preferred embodiment of the system of this invention, the Macintosh platform is considered to be the most appropriate.

While the ability to utilize COTS packages is essential, there are often severe limitations caused by the narrow scripting interface available between distinct applications. For this reason, it is far more desirable to incorporate functionality from existing object libraries providing a rich and complete API. Such commercial object libraries (as well as open-source code) are available to cover a wide range of techniques and capabilities. The need to integrate object-code libraries implies several constraints on the approach taken by the UCS environment as far as encapsulating blocks of compiled functionality (widgets). In particular, because such libraries are built on the underlying OS Toolbox, it is essential that the UCS threaded environment appear to such code as if it were within a stand-alone application. The principal impact of this requirement is on the need for a toolbox abstraction and patching layer, as well as the approach taken to providing a UI windowing environment. Since object libraries involving UI are unaware of the UCS and yet must be integrated into UCS windows, a number of otherwise viable approaches to providing a GUI environment

will not work. Given that changes to object libraries are not possible, the UCS GUI environment must take all steps necessary to ensure that non-UCS aware UI code, works un-modified within the UCS windowing environment. This UI sharing environment would preferably be implemented by associating dynamic and overlapping UI 'regions' with small executables such that the scheduling environment switches all UI parameters necessary whenever a given UI-related widget is running.

Security is obviously a major concern in most intelligence-related applications. Given the need to deliver reports and multimedia data to individuals, possibly beyond the confines of the system it is clear that reliance on security via access control alone (i.e., logging on to a Database) is not enough. Security must be built into the data itself. Given the nature of the intelligence cycle where the same item of data may be handled and annotated by many individuals, each of which may have different security privileges, we see that a sophisticated, data-centric approach to security must be supported by the environment.

The analytical process is frequently collaborative, that is it involves the need for multiple analysts to review each others work and interact with a given visualizer or display in order to discuss possible meanings for patterns found. For this reason, it is highly desirable that the UI for the UCS architecture inherently support collaboration such that users of the system residing on different machines can view and interact with a single display/portal in a coordinated manner, perhaps marking it up in a whiteboard-like manner as part of their discussions. Additionally, the ability to perform video- conferences during such sessions greatly enhances the utility of the environment. A system wherein an intelligence consumer can contact the analyst responsible for a given report and interact with both that analyst and the report is obviously far more useful than one that does not. This close interaction is critical to closing the intelligence system OODA loop (see below). Network level support for such conferencing and collaboration will be necessary.

On the subject of change, it is obvious that in any UCS connected to the external world, change is the norm, not the exception. The outside world does not stay still just to make it convenient for us to monitor it. Moreover, in any system involving multiple analysts with divergent requirements, even the data models and requirements of the system itself will be subject to continuous and pervasive change. By most estimates, more than 90% of the cost and time spent on software is devoted to maintenance and upgrade of the installed system to handle the inevitability of change.

Over and above the Bermuda Triangle effect, another software paradigm related phenomenon contributes to our inability to implement complex unconstrained systems. In object oriented programming (OOP) systems (the current wisdom), key emphasis is placed on the advantages of inheriting behaviors from ancestral classes. This removes the need for derived classes to implement basic methods of the class, allowing them to simply modify the methods as appropriate. This technique yields significant productivity improvements in small to medium sized systems, and is ideally suited to addressing some problem domains, notably the problem of constructing user interfaces. However, as size, complexity, and rate of environmental change are scaled beyond these limits, the OOP technique, rather than helping the situation, serves only to aggravate it. Because the implementation of an object becomes a non-localized phenomenon, tendrils of dependency are created between classes, and the ability of others to rapidly examine a piece of code during the maintenance and upgrade portion of the development (the bulk of the actual effort) is made more difficult. OOP systems generally introduce the concept of multiple inheritance to handle the fact that most real world objects are not exactly one kind of thing or another, but are rather mixtures of aspects of many classes. Unfortunately, multiple inheritance only makes the scaling problem worse. The maintainer is forced to examine and internalize the operation of all inherited classes before being able to understand the code and being sure that his change is correct. Worse than this, the 'right' change generally involves changes to the assumptions and implementation of some ancestral class, and this in turn often has a ripple effect on other descendent classes. Eventually, such systems max out at a level of complexity represented roughly by what can fit into a single programmer's brain. While this may be large, it is not large enough to address the complexity of a system for understanding world events, and thus an object oriented approach to attacking such a massive problem is essentially doomed to failure. OOP techniques still rely on the notion of one controlling top-down design. No such design exists in a complex UCS. Since we have said that change is fundamental to the nature of an unconstrained intelligence system, it is obvious that in addition to all the problems detailed above, we must also move to a totally new software paradigm and methodology if we are to succeed in this endeavor.

To summarize the principal issues that lead one to seek a new paradigm to address unconstrained systems, they are as follows:

a) Change is the norm. The incoming data formats and content will change. The needs and requirements of the analysts using the data will change, and this will be reflected not only in their demands of the UI to the system, but also in the data model and field set that is to be captured and stored by the system.

b) An unconstrained system can only sample from the flow going through the pipe that is our digital world. It is neither the source nor the destination for that flow, but simply a monitoring station attached to the pipe capable of selectively extracting data from the pipe as it passes by.

c) The system cannot 'control' the data that impinges on it. Indeed we must give up any idea that it is possible to 'control' the system that the data represents. All we can do is monitor and react to it. This step of giving up the idea of control is one of the hardest for most people, especially software engineers, to take. After all, we have all grown up to learn that software consists of a 'controlling' program which takes in inputs, performs certain predefined computations, and produces outputs. Every installed system we see out there complies with this world view, and yet it is obvious from the discussion above that this model can only hold true on a very localized level in a UCS. The flow of data through the system is really in control. It must trigger execution of code as appropriate depending on the nature of the data itself. That code must be localized and autonomous. It cannot cause or rely upon tendrils of dependency without eventually clogging up the pipe. The concept of data initiating control (or program) execution rather than the other way is alien to most programmers, and yet it becomes fundamental to addressing unconstrained systems. See patent ref. 11 for details.

d) We cannot in general predict what algorithms or approaches are appropriate to solving the problem of 'understanding the world', the problem is simply too complex. Once again we are thus forced away from our conventional approach of defining processing and interface requirements, and then breaking down the problem into successively smaller and smaller sub-problems. Again, it appears that this uncertainly forces us away from any idea of a 'control' based system and into a model where we must create a substrate through which data can flow and within which localized areas of control flow can be

triggered by the presence of certain data. The only practical approach to addressing such a system is to focus on the requirements and design of the substrate and trust that by facilitating the easy incorporation of new plug-in control flow based 'widgets' and their interface to data flowing through the substrate, it will be possible for those using the system to develop and 'evolve' it towards their needs. In essence, the users, knowingly or otherwise, must teach the system how they do what they do as a side effect of expressing their needs to it. Any more direct attempt to extract knowledge from analysts to achieve computability, has in the experience of the author been difficult, imprecise, and in the end contradictory and unworkable. No two analysts will agree completely on the meaning of a set of data, nor will they concur on the correct approach to extracting meaning from data in the first place. Because all such perspectives and techniques may have merit, the system must allow all to co-exist side by side, and to contribute, through a formalized substrate and protocol, to the meta-analysis that is the eventual system output. It is illustrative to note that the only successful example of a truly massive software environment is the Internet itself. This success was achieved by defining a rigid set of protocols (IP, HTML etc.) and then allowing Darwinian-like and unplanned development of autonomous but compliant systems to develop on top of the substrate. A similar approach is required in the design of unconstrained systems.

Any data substrate that is intended to model and understand the real world must, of necessity, imitate it in order to represent it. Just as for our own mental models, simulation must be an integral part of analysis in order to evaluate potentials. This immediately implies that some data can be artificial or predictive while other data may be 'real.' Both must be represented and behave identically within the environment. Furthermore, all data objects within the system must have the potential to have a spatial and temporal position. Many patterns evolve along the time axis and most 'events' involve, or are precipitated by, physical proximity in both space and time between the actors involved. This means that it must be possible to reconstruct the state of a captured datum at any point in time. Failure to embody this concept at the datum level would prevent the substrate from faithfully representing reality, and thus would involve the need to re-introduce complex control programs to supply this aspect. These control based edifices would naturally tend to diverge and thus leach

and/or dissipate utility out of the environment rendering it non-uniform and less useful as an interchange medium. A simulation in an unconstrained environment should just be an evolving set of data in which some portion (but not by any means all) is predictive or program generated. Once such artificial data outlives its utility, it must be easily purged from the environment to make way for a new simulation run. It is this failure to treat simulations as an integral part of a UCS that makes them so difficult to develop, and once developed, makes their results out of date, irrelevant and difficult to apply back to the real world. A well designed UCS architecture, in addition to all its other benefits, provides a means whereby simulations can become useful, relevant, and pervasive parts of the intelligence cycle (or indeed any application). This is a radical departure from current day simulation practice.

SUMMARY OF INVENTION

The present system and method meets each of these requirements and provides a robust and flexible system for storing, parsing, analyzing and typed data that is stored in a virtual ontological tree and is later available for retrieval from offline, nearline, or cache based storage and is viewed and processed in the language, interface and with the desired hyperlinks associated with the given User over a P2P or client-server architecture in a dynamic fashion and/or based on one or more user profiles. The issues presented herein are fully detailed in the patent application that have filed relating to the architecture described and attached hereto as appendices. This application details to the system level approach, in which each of these features are provided in a single UCS system.

The present invention provides the following:

1. A system for converting incoming unstructured data into a well described normalized form. Since the incoming data is multimedia and may represent some data type for which support is provided by the underlying OS platform, this normalized form include the ability to fully describe and manipulate arbitrarily complex native or non-native binary structures and collections. This support is provided by a dedicated 'mining' language tied intimately to the current system ontology (see appendices 6 and 7).

2. A system for accessing and manipulating data held either in memory or in persistent storage in its normalized binary form so that small executables, or 'widgets', within

the system can freely and effectively operate on data types they have never before encountered simply by knowledge of the 'type' of data involved (see appendix 4).

3. An 'ontology' or world model that represents and contains the items and fields necessary for the target system to perform its function. The ontology would preferably fully specify the form of the normalized binary data.

4. A memory system, tied to the ontology, which defines the structure of and access to any persistent storage containers that are required to contain the data.

5. A memory management system for splitting incoming data into those portions to be directed to each container.

6. A query system for querying each container to retrieve portions of such a composite object. Preferably, all database tables and queries are auto-generated from the ontology, thereby eliminating the role of the conventional Database Administrator (DBA).

7. A UI to display and interact with data within the system. In the preferred embodiment, the UI is automatically generated and its behaviors automatically handled by the underlying substrate thus removing this programming burden from the developer (thereby largely eliminating the role of the GUI programmer).

8. A memory system that forms collections of datums, and enables manipulation and exchange of these collections both within the local machine as well as across the network. In the preferred embodiment, such collections support the ability to attach arbitrary tags or annotations to the binary data they contain without in any way altering the binary representation itself. Additionally, the system supports the concept of either null or dirty (i.e., has been changed locally) datum.

9. The means (preferably implemented in software running on a processor) to specify, investigate and manipulate the inheritance of behaviors and fields from ancestral types described in the system ontology.

10. Support for incremental changes to the ontology and automated handling of the implementation and impact of those changes both on persistent storage as well as the UI and other dependant areas.

11. Inherent and pervasive support for the concept of units and their interchangeability. In other words, this system does not leave unit handling to the application logic. Such an approach would make it very difficult to meaningfully and easily exchange data.

For the purposes of this discussion, various appendices will be referenced and are fully incorporated herein. Each of these appendixes derscribe in detail one embodiment for the various pieces of the UCS system. As will be appreciated, various other functions and approaches could also be used.

The reader is referred to these lower level building-block patent applications as follows:

1) Appendix 1 – Flat Memory Model

2) Appendix 2 – Lexical Analyzer

3) Appendix 3 – Parser

4) Appendix 4 – Run-time type system

5) Appendix 5 – Collections

6) Appendix 6 – Ontology

7) Appendix 7 - MitoMine

8) Appendix 8 – User-centric Hyperlinks

9) Appendix 9 – User Interface Localization

10) Appendix 10 – Client/Server and MSS Architecture

11) Appendix 11 – Data-Flow

Process Flow and Related Issues

It is important to understand the intelligence process in more detail before attempting to describe the software architecture to address the problem. A conventional description of the intelligence process would lead one to define a system as a linear flow from inputs (feeds) to outputs (reports) having the following basic stages:

1) Capture

2) Storage, Retrieval & Indexing

3) Search & Monitoring

4) Analysis

5) Presentation

While this is a wholly inappropriate way to design a system, and does not reflect the reality of the intelligence process, nonetheless this breakdown gives us a useful framework in which to further examine some of the issues.

Capture

The main issue here is the large number of sources and types of data, each with its own unique requirements. Some of these sources and the associated issues are discussed below:

Video

The robust capture and use of video information presents one of the biggest challenges to a multimedia intelligence architecture. High quality video digitization, storage, and playback places the ultimate test on the server architecture and its associated mass storage subsystem. A great deal of external capture equipment is required including (but not limited to) satellite dishes, tuners, receivers (PAL, SECAM and NTSC – all variants), format converters, video switches, VCRs (multi-format), digitizers, CODECs, satellite tracking systems, de-scramblers, cable feeds etc. It is clear that the system must provide a framework for the definition, reconfiguration, and statusing of all the equipment connected to it. All equipment must be under automatic and transparent control of the system based on capture requests from the users. To this end, the system must provide some kind of TV guide capability with the ability to request programs of interest. Additionally, a 'snapshot' view showing all currently captured channels at the client workstations is required with the means to click on such a snapshot image and immediately request live view and/or capture of the material involved. Video (live or captured) must be streamed across the network to client workstations where it can be viewed and/or edited. This represents not only a massive

network load, but also due to the CPU intense nature of the capture, storage, and streaming process, it is clear that a video server cluster will require large numbers of machines to act in unison in order to support realistic client loads. Such a server architecture does not exist in the commercial space and thus must be developed and provided by the UCS architecture. Given a limited pool of equipment available for the capture process, and the differing costs of using a given equipment item to satisfy a user request, it is clear that the environment must provide some form of equipment scheduling capability which attempts to map present and future requests onto the available capture equipment by means of some kind of weighted graph. Equipment item usage cost is determined by how much the available stream capture capacity will be degraded by the use of that item. For example, many older satellites 'wobble' so these and other satellites require active tracking using a moveable dish. Most commercial satellites can be captured by fixed dishes. Assuming that a smaller number of mobile dishes exist than fixed, it is obvious that allocating one such dish to a given capture reduces remaining capacity far more than does the use of a fixed dish with multiple feed-horns and a splitter. The same effect is repeated through the equipment chain that must be created (e.g., format converters, switches etc.) in order to meet any given request. Capture equipment design and wiring needs to anticipate this problem and minimize this degradation effect. For example, use of a cable TV head-end to distribute captured video, removes the blocking implied by use of an analog switch to connect source to digitizer. This is a complex issue and must be closely coordinated with the system design and capabilities. Much equipment relating to video processing is not designed for computer control, and thus the system may have to provide the ability to control such equipment via IR links or whatever other means is provided. A generalized and fully programmable (from within the system) controller interface is required in this case. Massive storage capacity is needed to handle video. A key aspect of making use of video is to be able to determine what is being said during a given segment (e.g., a news report). There are a number of approaches to this problem, firstly, at least of a large number of NTSC transmissions, closed captioned text is provided and equipment is available to capture this. Since we wish to maintain the correspondence between a particular portion of a video and what is being said (to aid in search, retrieval, and playback), we can see that this text 'track' must be stored in parallel with, and using the same time code as, the video itself. The QuickTime™ architecture is ideal for this purpose, since it defines movies to be comprised of one or more tracks each of which can contain different media types. Thus the present system creates as an output to the

capture process a movie containing not only the video and sound tracks, but also a text track, and quite possibly later one or more voice-over tracks.

Text to speech, although in its infancy is another approach although this applies less well to foreign languages. The choice of video CODEC is determined by the quality required as well as by the need for real-time symmetric capture and playback, preferably using CPU resources alone, not dedicated cards (which rapidly become obsolete). Storage of multiple video resolutions can significantly reduce the required server resources. Video sources, especially those derived from terrestrial transmissions, must be captured locally, thus it is clear that a 'logical' video subsystem is likely to be physically distributed, possibly globally. Given the streaming nature of video, this implies a number of other challenges relating to streaming, load balancing, and storage. The UCS architecture must support mechanisms whereby all these requirements can be tailored and handled. Much of the video captured (especially in PAL and SECAM formats) will not have a text track and therefore a key aspect of video capture (and indeed any multimedia capture) is the ability to 'tag' the video with other related items (such as news stories) which are more easily associated. The environment must support arbitrary tagging of any datum with any other datum(s) in order to render it 'computable'. A distributed video server and client(s), video snapshot server and client(s), equipment server and client(s), and various other video related technology have been fully implemented based on the technologies revealed in the referenced patents, particularly patent ref. 10. The details of these implementations and some of the unique features involved will be fully revealed in future patents.

News Feeds

News stories and reports form one of the most useful, timely, and easily leveraged forms of open-source feed. News feeds are available in many languages and come in both localized (national) and global varieties. Examples are Reuters, API, BBC etc. Feeds are delivered in a variety of ways including satellite downlinks, analog land-lines, Internet sites, dial-up access, and CD-ROM based delivery. Archival news feeds are usually available for purchase from the publishers although delivery media can be archaic. There is little standardization in format between the feeds although an XML standard for Internet delivery is in its infancy. Multilingual issues abound and normalization can be quite a challenge. Many local feeds have poor quality control over syntactic structure. News feeds are characterized by a relatively low bandwidth with a high semantic content. Storage issues are

minimal. For these reasons, the present system provides a news server based on the technologies revealed in appendix 7 and appendix 10 has been fully implemented under the system of this invention.

PhotoWire Feeds

Photowire feeds are available from many of the same global sources as are news feeds, and delivery platforms span a similar range. Images come in a huge variety of standard (and not so standard) formats and the system must natively handle all of these, or at a minimum convert losslessly to one of them. Images can be quite large and an associated mass storage subsystem is required. Unlike video, isochronous delivery to the client is not required. The concept of an image preview or 'picon' is key to ensuring that full image retrieval is only required for analysis or editing. Images from these sources can form a powerful part of any multimedia presentation. Many sources of photowires also provide graphics and illustrations which are intended for use in publications supported by the feed. These graphics (e.g., stock charts, topical maps, etc.) can be very helpful in understanding issues and in presenting conclusions. Support for the capture, storage, and retrieval/use of these graphics must also be provided by the environment. Graphic formats are generally different from image formats since they are intended to allow editing of the graphic for incorporation into page-layout and similar applications. The Adobe Illustrator™ format appears to be the most widespread. An Image server based on the technology revealed in patent reference 10 and which is capable of handling all image types discussed herein, has been fully implemented under the system of this invention.

Satellite Imagery

Satellite Imagery is an important part of the intelligence process. Satellite images are essentially just high resolution images which contain additional semantic meaning by virtue of the fact that the 'where' for the image can be computed by knowledge of the satellite parameters and position involved. Thus it is clear that there is a close tie-in between satellite imagery, and the mapping and GIS facility that must be provided by the environment. The environment must be able to automatically project/overlay the image with respect to a map background so that the information it contains can be related back to other data in the system. Satellite images generally contain multiple 'bands' of data for different frequencies and sensors, and these bands can be used or combined to extract additional knowledge regarding

the contents of the image. Tools for this purpose must be provided. Commercial satellite imagery comes from a variety of sources including weather satellites, LandSat, SPOT etc. Delivery mechanisms for some (e.g., weather) involve the use of receiving dishes. For others, the imagery is delivered on a variety of media (often tape) or by FTP download. For the most part, satellite imagery is a non-real-time feed. Government agencies may have access to a number of other forms of satellite imagery whose nature and content is not discussed herein.

Specialized Imagery

Particular applications may require support for other specialized forms of imagery with additional semantic meaning. Examples include fingerprints, identification, x-ray images, astronomy, etc. Each of these types essentially requires its own server subsystem to provide extraction and support for the additional semantics. The environment provides for the easy creation of such servers. Most such sources will require a connection to some external equipment or system to provide capture and possibly storage and search of the imagery. In all other ways however, such subsystems are similar to the generic imagery subsystem.

Sounds

Like video, recorded sound can convey a richness and subtlety far beyond that possible with other media types. Because video often includes sound, there is an obvious overlap between the two data types. Sounds come in a number of formats and have widely varying quality levels. Like video, sound must be delivered isochronously to the client, however, data rates are significantly lower though still high enough to require a clustered server and associated mass storage subsystem. Sound sources include phone recordings, covert intercepts, and published media. Like video, a key consideration with sound in order to attain computability, is the ability to convert it into one or more associated text tracks. For this reason, the sound architecture of the present system, like video, uses a time based media framework such as QuickTime™. As with video, voice-overs (or translations) are supported as distinct tracks. Text tracks are, in parallel, routed to the text subsystem to allow associative search. A Sound server based on the technology revealed in referenced patent 10 is the preferred embodiment of such a server.

Internet

This source is perhaps the most widespread and the easiest to capture of any of the sources described. Unfortunately, with the exception of a few trusted sites, it is also one of the lowest grade and most misleading sources on which to base any automated calculations. Techniques to crawl or spider the web are widespread and readily available, often built into the underlying OS (e.g., the Macintosh 'Sherlock' facility), and because it is web data (i.e., HTML or even better tagged XML) it is designed to facilitate easy capture and use by digital systems. The web contains many invaluable trusted sources for real time data such as news, stock feeds, weather etc. and provided one sticks to these, it forms a key part of monitoring what is going on in the world. The rest of the web data, i.e., the un-trusted bulk of it, must be treated with skepticism much in the manner needed for a covert intercept. That is a 'discriminator' phase is required to determine usefulness and relevance. This having been said, much valuable insight can be obtained from such data, especially if one includes e-mail capture into the equation. Storage requirements for web capture are relatively manageable, and like news feeds it is characterized by high semantic content (once filtered). The key issue for any secure installation, is that mining the web on an automated basis implies a connection between the system and the web itself. This is dangerous and often totally unacceptable, especially in government installations. For this reason, the system provides the ability to control a 'drone' insecure capture capability which then uploads its finds, via a secure path, to the system itself (which may not be physically connected to the web in any way). Such an Internet server based is prefereable based on the technology disclosed in appendix 7 and appendix 10.

Published Data Sources

Perhaps the highest grade and most reliable of all non-covert sources, published data also comprises the largest single source of any described. There are literally tens of thousands of different database and information publishers, each specializing in particular areas. The total amount of data available is immeasurably larger than the total content of the Internet. Few publishers post any high grade data on the web due to the lack of a business model to do so. Many that have done so have now gone out of business and this process is on-going. Because the livelihood of such sources is predicated on their continuing completeness and quality, published data provides some of the best supplies of background information necessary to populate a system's 'lens' of understanding. Published data sources

come in many forms and tend to be expensive. CD-ROMs are now becoming the dominant distribution media although on-line databases such as Lexus/Nexus contain vast amounts of information that can be easily accessed and incorporated into the environment.

The extraction of information from these sources tends to be a non-real-time batch process and requires a parsing process that can parse data on a per-source basis. Because publishers have no interest in facilitating the automated extraction of their intellectual property, this data tends to be in semi-structured formats with all kinds of inconsistent usage, even within the same data source. On-line sources tend to have built-in defenses against automated mining. To extract useful normalized data from these sources therefore, the present invention provides a very powerful, generalized, and robust data mining framework tied to the system data models. The ability to rapidly absorb a new published source and seamlessly integrate it into the system enables the system to react in a focused and informed manner to on-going events. When a particular new issue suddenly becomes critical, as they always do, it is likely that very little information exists in the system on the subject. To empower the analysts to rapidly come up to speed on the issue and make analyses relating to it, the system provides a turnaround time measured in hours or at the most days, to acquire and integrate new published sources. Classic mining techniques and system architectures cannot meet this requirement. The preferred technology for enabling this aspect of the system is described in Appendix 7.

Legacy Systems

All large organizations utilize as part of their operations a number of 'legacy' information processing environments both internal and external. Much of what an organization is, has, and knows is encapsulated in these systems. Such legacy systems do not go away, and often tend to be based on old or antiquated equipment. The present system makes use of the information contained within these systems as part of it's operation. Generally such legacy systems present themselves as databases, usually relational. The ability to access, mine, and source/sink data to/from these legacy systems is often essential to system operation. More specifically, the architecture provides a generalized framework for interfacing to and using such systems through the specification of 'scripts' utilized via an encapsulating UCS server. Ideally, the implementation of a connection to such a legacy system would involve little more than definition of the necessary logical scripts. The SQL language makes this relatively easy although it is often the case that custom code is required

in order to implement such a connection. As the, UCS architecture also provides the means whereby plug-in modules, defined on a per application, per legacy system basis, can be registered within a standard UCS server. In legacy systems, external containers may also be grouped by providing customized functionality specific to a given data type. Thus for example, a connection to a fingerprint recognition system would be treated as a legacy system requiring an encapsulating UCS server. The system and methods disclosed in Appendix 7 and Appendix 10 are sufficient to implement such a custom legacy interfaces.

Manual Data Entry

In certain cases, this may be the only practical means of capturing data, especially data that does not yet exist in the digital domain. The UCS environment also supports the ability to perform manual data entry based on a system ontology. One refinement of this is the provision of a programmable UI scripting capability to provide for the possibility that a process can be written to obtain the data somehow, and enter it not by ontology based mining, but rather by scripted data entry. Once any data (manually entered or otherwise) is in the system, it is also possible to edit and change it and thus the auto-generated UI to the system supports data entry, complete with some level of validity checking, based directly on the system ontology definitions. The preferred ontological framework of the present invention is described in Appendix 6.

Documents

Much textual data exists in the form of word processing documents and this is a legitimate source of data for the system. Word processing documents are generally not just simply plain text, but rather contain embedded formatting and style information mixed in with the actual content. These formats are often proprietary. The final appearance of the document may have more information content to it than would be represented by the textual content alone, and for this reason a compliant system must have the ability to store and retrieve these documents in their original form, possibly for additional modification using the appropriate COTS application. Text held in these proprietary formats may not be directly useable for system functions. For these reasons, the system is able to strip the plain text content out of such documents and normalize it. The existence of scriptable COTS applications, capable of import/export of a variety of text formats makes this practical by creating UCS wrapper servers that script such applications, extract the normalized

information by scripting COTS applications (or by dedicated plug-in code), and store/retrieve the full document contents as required. Some of the more common formats include PDF, Word, RTF and others. See appendix 7 for further details of this aspect of the system.

Maps

Full support for the capture, visualization, and creation of maps is also provided by the system. Sources of such mapping data include such government agencies as NIMA, USGS, the US Census and others. Custom specialize maps are often created by dedicated COTS mapping environments. Such environments generally support import/export to/from a number of standard map interchange formats and the UCS map support also includes the ability to input and output from/to some number of such formats. In the case of more global and extensive data such as that from government agencies, the system provides the inherent ability to mine and normalize such data for system mapping purposes. NIMA maps can be obtained for the entire world on CD-ROM sets formatted according to MIL-STD-2407 (Vector map 0 and 1) and the ability to mine and interpret this format is basic to system operation. Targa and similar data are also be natively supported. Detailed world maps require significant amounts of storage at the map server(s) but not more than can be accommodated on the large disks (or raid arrays) available today. Speed of random access to the data stored on these disks is absolutely critical to map server rendering performance and in the most demanding situations, budget permitting, massive fronting RAM disks and preferably also large amounts of system RAM at the server (to allow data internalization) will be required. A compliant map and GIS server is preferably based upon the technology described in Appendix 5 and Appendix 10.

Covert Digital Intercepts

Few organizations outside government intelligence agencies have the resources or legal rights to engage in this kind of activity. For this reason, let us assume the existence of equipment and systems capable of taking a digital stream off a satellite or 'tapped' communications path, de-multiplexing it into its constituent parts, and delivering those parts to the intelligence system either as text or standard multimedia data. A number of significant issues occur once the source of data is an intercept, and these need to be anticipated by the architecture. Firstly, the syntactic and semantic quality of the data is likely to be much lower than for other forms of capture. This is partly because the data was not intended for capture,

but also because the de-multiplexing and re-assembly processes will be less than perfect and so some of the data may be partial, corrupt, or unusable. This implies a far greater burden on the robustness of the process used to convert data into its normalized form. If the approach taken is to 'parse' the input in some manner, it now becomes essential that the parser have error recovery and fallback strategies, rather than simply aborting following a syntax error. In this manner, it remains possible to extract and possibly use those portions of the item that are valid while retaining corrupt portions for possible subsequent interpretation by human beings or other processes in the environment. The variety of forms that are likely to be encountered in covert intercepts is significantly greater than for most other feeds and as a result the present invention provides a robust mechanism to decide 'what' a given item represents prior to invoking a parser or parsers to attempt to normalize it. Generally with other feeds, this identification phase is relatively simple. With non-covert feeds (other than the Internet), it is frequently the case that all or most incoming data is captured to persistent storage. With covert feeds, this is seldom the case. Much of the content of a covert feed may be irrelevant, thus the system provides an additional 'phase' in the capture process that is responsible for determining if the item should be kept or discarded. This determination is preferably under the control of the analysts using the system and the specific algorithm used will differ between analysts, data types, and over time. This 'discriminator' phase is closely tied with the concept of 'Interest Profiles' or alerts defined by the analysts and running autonomously in the system servers. See referenced appendix 7 and appendix 10 for details on the technology that is preferably used to implement this functionality.

Others

There are of course an almost infinite number of other possible media types and sources. Examples might include seismic data, monitoring systems of all kinds, stock feeds, scientific experiments etc. The intrinsic ability to add these data types to the ontology and rapidly implement an encapsulating server(s) for acquisition, search and retrieval, is fundamental to the present invention.

Storage, Retrieval & Indexing

The issue of storage and the strategies necessary to effectively index items in storage for rapid retrieval takes on a whole new level of complexity. The main problem is that each different multimedia type implies a different storage and indexing requirement. This means

that the conventional approach, i.e., store everything in a relational database system (RDBMS), does not work well.

RDBMS storage is essentially based on the use of grids or matrices to store information. Because each cell in the matrix has a known size, efficient indexed access is possible. An RDBMS system is therefore best suited to the storage, search, and retrieval of small fixed sized fields, especially those that are numeric. For this reason in a UCS environment, RDBMS storage makes most sense when applied to these kinds of fields, not to large text fields or multimedia content. More specifically, because storage is distributed across a number of dissimilar 'containers' of which a RDBMS/SQL container is just one, it is clear that in order to re-assemble a complete multimedia item for display, we need a common unique ID number that can the applied to all containers to retrieve content for an item (see patent ref. 6). The RDBMS system is ideal for defining these ID numbers and retrieving the basic fixed sized fields of an item. In the preferred embodiment, RDBMS data tends to be relatively small, and generally fits easily onto a single large disk.

Variable sized text fields are best stored and searched via an inverted-file text engine. In the inverted file approach, for each significant word in the dictionary, the inverted file stores a list of all documents containing that word and the position(s) of that word within the document. Search and retrieval in this system therefore occurs via the inverted file list which is far more efficient than the corresponding brute force keyword scan in an RDBMS. Additionally, because of the inverted file organization, statistical word relationships can be built up from the full set of data in the system and this allows powerful concept type searches which are poorly supported under RDBMS systems. Text stored in an inverted file container tends to be moderately large and may require a RAID array. Furthermore, the inverted file itself is generally best placed on a separate fast disk (array) preferably fronted by a large RAM disk/cache to increase search and query performance (see appendix 10 for additional details).

Video information requires storage capacities many orders of magnitude larger than those described above. Terabyte or petabyte capacities are not uncommon. In addition, the nature of video is that it must be delivered to the client as an isochronous (i.e., constant data rate) stream at a relatively high bandwidth. Furthermore, the CPU load represented by the actual streaming process is considerable, and thus conventional desktop computers are capable of delivering only a small number of high quality video streams at a time. Another

key aspect of video is that any given video segment contains a time axis and thus to find and view a relevant portion of the video the ability to tie searchable/indexed information to this time axis is required. For all these reasons, video probably represents the worst case scenario for any UCS storage, indexing and delivery architecture. To address the storage capacity, the present system supports robotic autoloader mass storage using fast random-access media (to minimize wait time to start a play). Media types like CD-ROM and DVD are a natural match. Obviously because these media types have limited sustained data-rates by comparison with fast disk, but more importantly have a relatively long 'seek' period, it is not practical to sustain multiple streams from a single such disk. For this reason, the system also provides automatic disk caching during playback and supports large numbers of media drives into any given area of robotic storage and media duplication. Automated, unattended 'burning' of media and migration from capture cache is also provided and is preferably implemented. Finally, because of the CPU load and the need for isochronous playback, the video server is implemented as a large cluster of machines tightly integrated with the robotic storage so that the 'master' machine can select a 'drone' machine on the basis of current loading (or otherwise), load the media into a drive connected to that drone, and then commands the drone to perform playback. See patent appendix 10 for additional details. Indexing implications have been discussed previously under "Capture" above.

Image data can be relatively large and generally requires a robotic autoloader component, however, unlike the video case, there is no isochronous requirement (since image files can be 'downloaded' entirely when accessed) and the need for a large image cluster is reduced. As a result, in the preferred embodiment, the image storage consists of a low resolution 'picon', accessible immediately from server disk storage. This is then combined with a high resolution full image which may require robotic access to retrieve. Many client uses of images can be handled using the picon alone thus avoiding excessive robotic accesses. Indexing in the case of images is straightforward since they are simply referenced via the common unique ID shared between all containers (see appendix6 and appendix 10).

The storage requirements for Maps have been discussed previously under "Capture". Map indexing is totally different form all other forms above in that it is spatial, that is that the map is accessed mainly by spatial position. Unlike other data types described above, maps can be constructed on-the-fly from a map database, and thus the map container is capable of responding to map requests without the need for an 'id'. Specialized maps can also be saved

and then referenced, and in this case the unique 'overlays' that customize the 'default' base map overlays are probably best be stored either in the RDBMS container or in other ontology derived storage along with details of the map projection, scale, and other legend elements.

The Internet presents another unique storage situation. In the case of the Internet, indexing is via URL, and the storage device is the Internet itself. Nonetheless, this variant is transparently fitted into the same abstraction as all others described above. Other data types may imply yet more variants of the storage and indexing problem.

It should be noted that the product of many feeds to the system is not a single type as discussed above, but rather some combination of multimedia parts each of which must be routed to the appropriate container but tied back to each other by use of a common unique ID. This dispersal aspect is further discussed in Appendix 6.

Search & Monitoring

One of the primary issues with searching over multiple dissimilar 'containers' is the need to create a framework within which the necessary search plug-ins can be registered with the environment and the corresponding GUI necessary to easily specify such a search can be tied-in to match. As described above, each container presents a different set of search capabilities varying from standard SQL and text searches to such things as voice and image recognition.

The present system provides a two-layer approach to querying and query specification. The lower layer represents the registered search capabilities of each specific container. The 'language' supported by this lower layer is completely open ended in order to permit new media types and search engines to be easily added to the environment. The result of a search conducted at the lower layer is a list of 'hits' (i.e., unique ID, together with relevance and other details if appropriate) that is then passed to the upper query layer. This upper layer has a well defined and preferably limited language, the primary purpose of which is to specify logical combinations of the hit-list results returned by the lower layer modules. Thus the language contains such Boolean operations as AND, OR and NOT. In addition, to support query optimization based on knowledge of the query domain, operators like AND THEN are also supported. The AND THEN operator implies that the query appearing before the operator is performed first and the resulting hit-list is then passed along with the query appearing after the operator. This allows efficient pruning of the search space in the

container(s) implementing the second portion of the query. Other operators that would preferably be supported at the upper level include such things as MAX (limit # of hits returned), RELEVANCE (limit relevance returned), ORDER BY, GROUP BY etc. Further details of a system that can provided this functionality is set forth in Appendix 6.

In the preferred embodiment, a querying GUI whose outermost aspect relates to the upper query layer, and within which specialized UI 'pages' can be displayed in order to specify container specific lower level queries is provided. The nature of these UI plug-in modules for well known querying engines such as SQL or inverted text files is fairly straightforward. When the list is broadened to sounds, videos, images, maps etc., however, the variety of UI components embedded within the querying interface in a unified manner becomes quite large. As such, querying and selection via visualizers is tied into the present invention.

Examples of plug-in search engines (accessed via corresponding GUI) include:

a) **SQL** – basic numerical, date, range, keyword, Boolean etc. search criteria.

b) **Text** – statistical relatedness, stemming, proximity, multilingual, fuzzy and concept searches.

c) **Images** – Face recognition, pattern recognition, fingerprints, clustered and similar searches.

d) **Video** – Searches based on text track, voice recognition, scene analysis, close caption etc.

e) **Maps** – topological queries (within, next to, etc.), spatial relationships, terrain features, range, distances, routes, measured paths etc.

As to the issue of monitoring new inputs to the system for compliance with certain criteria, this can be treated as simply an automated query applied to new input. For example, a multi-container query can be defined that returns only those hits that meet our desired criteria and then launches this query into the system to be automatically applied to all new input. This type of automated query will be referred to as an "Interest Profile" (see Appendix 10). The benefits of the two layered query approach now becomes clear because this same

mechanism may be applied by combining the 'hits' from parts of an interest profile in order to determine if a globally compliant 'hit' has occurred.

Unfortunately, the business of monitoring new inputs can be considerably more complicated because of the fact that not all algorithms to define a 'match' can be expressed directly to the querying layer. Often, to determine a match the analyst may need to combine a number of different functions. For this reason, the system provides 'widgets', each of which is capable of performing part of the analysis using whatever techniques are appropriate. This means that in addition to distributed queries in the querying language, widgets are preferably distributed that form part of the matching algorithm. The system of the present invention allows as large a range of widgets as possible to be used in defining these analyses. As such, the system provides a distributed framework whereby arbitrary algorithms expressed either as searches or via widget wiring can be placed into the input pipe of the UCS and can result in automated notification of the analyst when the desired match is found. See appendix 10 and 11 for additional details.

Notification to the analyst may be as simple as beeping (or speaking) at his terminal and maintaining a list of pending hits to be viewed. Alternatively, notification could be handled via automated e-mail delivery. Finally, the present invention supports the ability to initiate execution of arbitrary widgets supplied by the user to perform whatever action in necessary when a match occurs. By using this facility, the system can now trigger automated but targeted responses to the occurrence of any given situation. Obviously the nature and scale of these responses is limited only by the imagination of those configuring a particular UCS system. See appendix 10 for details.

Analysis

The thrust of this invention is the infrastructure and architecture necessary to support any combination of analytical tools, and to allow those tools to interact between each other over a common substrate. There are literally thousands of effective analytical tools out there, most of them operating in splendid 'stovepipe' isolation, some small fraction of them available as COTS applications. Such tools can be integrated into a UCS and used in conjunction with others which, in combination with the other features provided by the present invention, can be used with devastating effect. The only 'analytical tools' that would preferably be built in to any UCS is a suite of visualizers, the basic querying tools, and the

ability to "wire" these tools and others together into ever more elaborate domain specific algorithms. The UCS architecture preferably facilitates and captures this process using the system and method disclosed in Appendix 11.

Presentation

As discussed previously, the final stage of the intelligence process is to deliver analyses to the intelligence consumer in a form that is multimedia rich, and which can allow that consumer to interact with the analysis in order to examine assumptions and determine if more information is needed. Reports must themselves be active and interactive custom portals relating to a given subject. The creation of such reports must be made easy enough that analysts themselves can accomplish this step. More importantly, reports are not static, that is, once an intelligence consumers needs are sufficiently well understood and algorithms designed to meet those needs have been expressed, it is essential that the system be able to deliver 'today's report on...' to the consumer on an automated basis with no further analyst involvement. This trend is already being seen in web portals that allow limited customization on a per user basis. Obviously, an intelligence system must take this approach to a whole new level. As mentioned previously certain end users will require a simplified 'executive' interface and the present invention provides such an interface. A goal, at least for some consumers, is to allow them to directly express their own interest profiles and to have these (as well as those from analyst initiated profiles) appear in their portals immediately any 'hit' occurs. This closes the intelligence OODA loop (see below) and allows the consumer to determine what additional analyses he needs in a much more timely manner. Through this approach the system can manage the information overload problem that is experienced by the intelligence consumer himself, not just that of the intelligence professionals he tasks. See appendix 10 and 11 for details.

The Intelligence Cycle

In the traditional intelligence cycle, the intelligence consumers make known their needs for information via requests that are passed to the organization that assigns priorities to information requirements. Determination of priorities leads to tasking which results in the various collection mechanisms or agencies taking steps to gather the raw information necessary to pass on to the analysts. After performing whatever analyses best fit the problem

domain, the analysts prepare reports, which are then reviewed and coordinated and finally disseminated back to the original intelligence consumer.

The cycle described above represents the best thinking on how intelligence should work from the 1940's and 1950's. The cycle is still utilized today by the government intelligence community. In today's fast moving and information rich environment, such a cycle is unfortunately inadequate to the task of tracking the complexities of unfolding world events. A full description of the problems with such a cycle is beyond the scope of this document, however, the basic problems can be summarized as follows:

a) The cycle is too slow. Indeed it is not clear that it is a cycle at all, since most requests result in just one iteration. The existence of various organizations/bureaucracies in the cycle combined with the time taken for information to pass through the bureaucratic interfaces in the loop mean that the cycle cannot keep up with evolving events.

b) Because it is essentially command driven, the cycle only allows looking into questions that the intelligence consumer already 'knows' to ask. As discussed previously, the reality is that the cycle must support the discovery of things you didn't even know were important. The September 11th attacks provide a perfect example. This top-down approach may have suited a situation where the enemy was known and stable (i.e., USSR), but it does not deal well with today's world where enemies are small, distributed, loosely coupled, change constantly, and can have impacts disproportionate to their size. The intelligence consumer cannot anticipate all possible threats and task the complete cycle to investigate each.

c) The lack of feedback in the cycle between the consumer and the analyst, combined with the inability of the consumer to directly access and examine the backup material leading to analytical conclusions, tends to create a situation where the final product may not meet the consumer's requirements and thus redundant iterations through the cycle with corresponding increases in time and cost are required.

Modern competitive and business intelligence cycles are now based on some derivative of the Boyd cycle (or OODA loop). This cycle was developed by Colonel John

Boyd as a result of his studies (and experience) of air-to-air combat in the Korean war. What Boyd discovered was that the main factors that enabled US pilots to consistently win dogfights, were firstly that their F-86 fighter aircraft's canopy was larger than that of the opposing Mig-15's, thus giving a greater field of vision, and secondly, that although the F-86 aircraft was larger and slower, it was more maneuverable (higher roll-rate) thus allowing US pilots to make more frequent adjustments. Boyd was later largely responsible for the design of the F-15 canopy and perhaps more than anyone else, contributed to development and deployment of the F-16. The result of formalizing and abstracting Boyd's insight became a fundamental part of air-force tactics and later of military tactics in general.

The central idea behind the OODA loop is that all thinking entities are executing OODA loops of their own (consciously or otherwise), the key to success in any conflict or competition is therefore either:

a) Being able to cycle around the loop faster than your opponent.

b) Disrupt the opponents OODA loop to cause him to slow down or make mistakes.

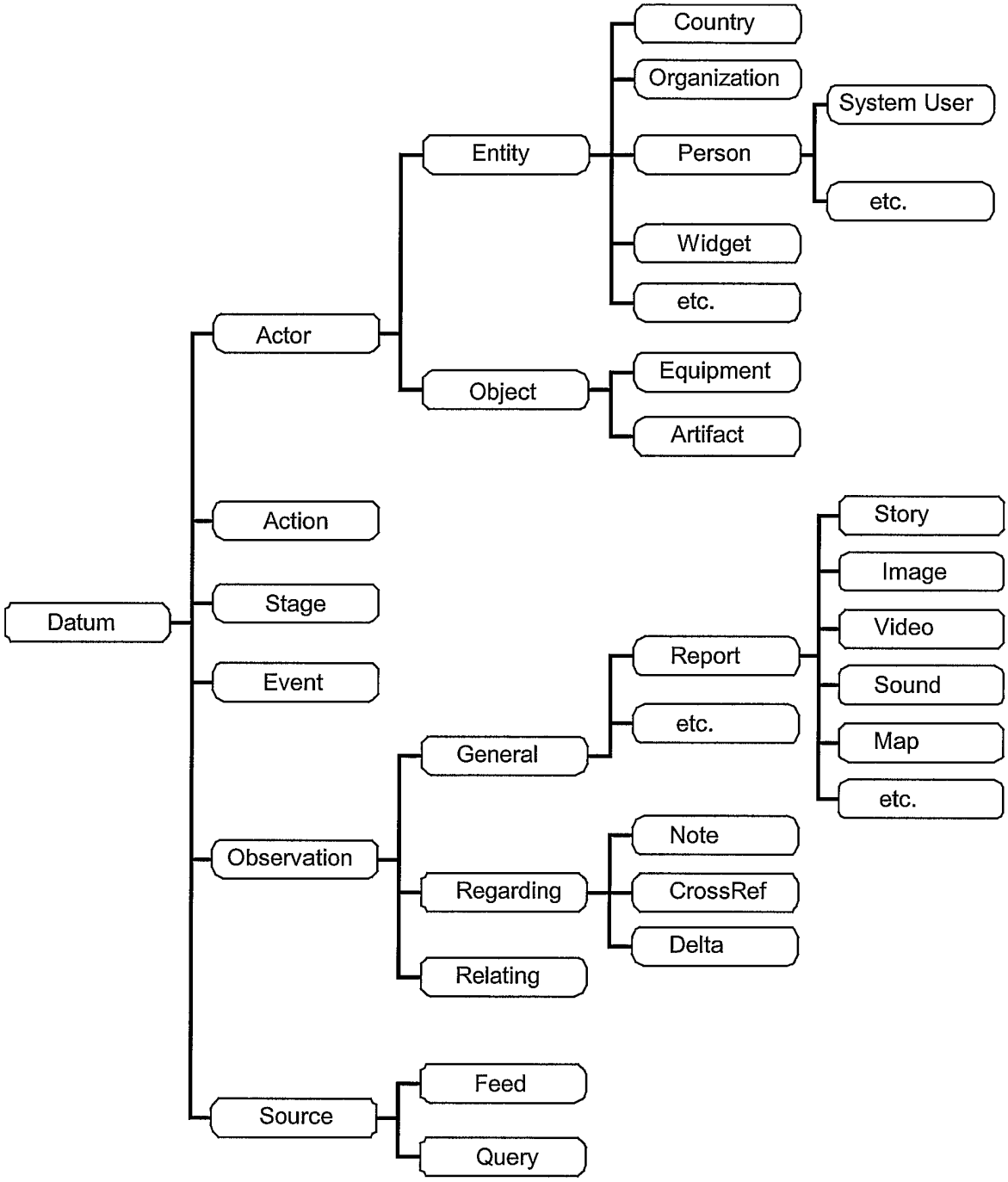
c) Alter the tempo and rhythms of your own loop so that the opponent cannot keep up with you.

For a full description of the OODA loop and how it ties in with the intelligence problem, as well as a complete bibliography in this area, see the paper **"Avoiding Information Overload Through the Understanding of OODA Loops, A Cognitive Hierarchy and Object-Oriented Analysis and Design"** by Dr. R.J. Curts, CDR, USN (Ret.), and Dr. D.E. Campbell, LCDR, USNR-R(Ret.). This paper can be downloaded from www.belisarius.com. This site deals with business intelligence and is heavily focused on the work of Boyd. While this author is not in complete agreement with the paper's assertion that object oriented (OO) techniques provide a practical approach to addressing the issue, the paper does effectively describe the need for a ground-up approach, and a consistent method for representing and storing data.

For this reason, the intelligence cycle itself needs to become a Boyd cycle. The speed with which it is possible to iterate through the loop is critical to success. Moreover, this same OODA loop would preferably be practiced at all levels of the intelligence hierarchy. This

need for rapid iteration and recursive loop cycling is a key driver for the end-to-end UCS approach described in this document. By using the present system, the barriers between intelligence consumers and those involved in the intelligence process itself can be broken down, and the rapid feedback loop required can be implemented. Most importantly however, the key lesson of Boyd's teachings is that the ability to rapidly adapt to change is the single most important determinant in any competitive situation. The present system provides a data-flow system that is driven entirely off ontology, allowing almost instantaneous modification and adaptation to changes in the environment. No other approach currently offers this capability, and thus, no other current approach stands any chance of addressing today's critical need in the intelligence community.

A High-Level Intelligence Ontology



The ontology presented above is an example high-level ontology targeted at intelligence. This is an example and in no way should such an ontology be mandated by the system architecture. A full discussion of this example ontology is given in Appendix 6. For the purpose of deriving some level of meaning from incoming observations, the application of such an ontology can be summarized as follows:

1) Over time, or by pre-loading from published or legacy sources, the system builds up a set of known actors that can be identified by name (or alias) in new input. In addition, the ontology for actions must be populated. At the same time, system input sources are identified and the necessary scripts to convert the contents of those sources into the normalized system ontology (primarily as observations) are developed.

2) Once the stream of observations from feeds is underway, the dictionary of actors and actions can be used to identify which data in the system an observation relates to (i.e., the actors involved), and the kinds of interactions that are occurring between those data (actions). Over time, the system builds up statistics on the relations between various elements of the ontology.

3) Analysts define conceptual axes to the system together with the algorithms necessary to compute axis intercepts. These conceptual axes can now be used to re-cast the data in the system in a new light, looking for trends, relationships and anomalies.

4) Analysts build models for the motives of various entities and to define algorithms for mapping between motives and the actions available to those entities. This allows modeling and prediction to be used as part of the matching process in the input stream. More importantly, system data can now be re-cast and visualized in light of the motive-action models in order to look for patterns in the data that significantly correlate with meeting the motives of specific entities of interest. Since entities rarely announce their intentions beforehand, this ability to interpret incoming data in terms of how it maps to entity motive models is key to finding insights to answer the 'who' and 'why' questions.

5) The process of 'event reconstruction' also occurs. That is, given the observations the system receives, knowledge of the actors involved and models of those actors motives and available action space, the system is able to perform a surface-tension type analysis looking for explanations of the event described that most closely match the motives of one or more of the initiating (i.e., subject, not object) actors involved. By postulating that this is in fact what occurred in the event, it becomes possible to define a pattern in the observations leading up to the event that represent an indicator that a given entity, or entities, are attempting to cause a similar event to occur. Much of this process involves the analyst using the various visualization tools. Alternatively, however, the process can be automated as the analyst expresses the algorithms he believes imply a given motive vector is occurring.

6) Examination/visualization of 'instrumented' events occurring over a period of time against entity-motive models allow the system to reveal trends, patterns, and anomalies in those events. This in turn yields the possibility of identifying hidden entity involvement, known entity 'meta-intent', and ultimately in using that knowledge to predict future behavior. Once future behavior can be predicted to some level of accuracy, the system can allow the intelligence consumer to move from a reactive to a proactive role in order to influence the occurrence (or non-occurrence) of that behavior. Once this point has been reached, the system allows the Boyd-cycle described in the previous section to be iterated over more quickly and thus gives the intelligence consumer a significant advantage over others, this is of course the ultimate goal of any intelligence system.

To present these ontology ideas in a more graphical and perhaps more intuitive way, think of the problem as though it were a particle-physics experiment occurring within an accelerator. In this example, suppose the experiment consists of a target into which is fired a particle beam. The collisions between the beam and the target produce events which emit a set of secondary particles which may be observed using different sensor devices each designed to detect a particular particle type. The data streams resulting from each sensor are fed into a computer for recording and subsequent analysis. Since it is likely that not all particles resulting from the collision are detected, the purpose of the analysis is to use the

data gathered to infer exactly what type of event must have occurred during the collision and from that to deduce the nature and behavior of the particles involved. The next stage is then to use this model to predict other events and then search for the signatures of those events in order to confirm the model.

In an intelligence system the situation is very similar although the terminology changes. A number of sensors and other data capture devices capture aspects of an event (or future event). The goal of the system is still to reconstruct what event has occurred by analysis of the observation data streams coming from the various feeds. The variety of feed and sensor types is infinitely larger than in the particle physics case, however, as for the particle physics case, many effects of the event are not observed. The major difference between the two systems is simply the fact that in the intelligence system, the concept of an event is distributed over time and detectable particles are emitted a long time before what is considered "the event". This is simply because the interacting 'particles' are intelligent entities, for which a characteristic is forward planning, and which as a result give off 'signals' that can be analyzed via a UCS in order to determine intent. In the recent September 11th attacks, for example, there were a number of prior indicators (e.g., flight training school attendance) that were consistent with the fact that such an event was likely to happen in the future. The intelligence community failed to recognize the emerging pattern, however, due to the magnitude of the search, correlation, and analysis task. This is exactly the issue addressed using the UCS of the present invention combined with a domain specific ontology and the other capabilities.

From the discussion above, it is clear that a radically different approach is needed to solving the problem of unconstrained systems. The architecture of the present invention is based on the concept of a distributed data-flow driven environment, rather than a conventional control-flow based solution. The form, content, and behavior of the data in the environment is described via an ontology that is specific to the given application. Control and/or data flow based programs (known as widgets) are caused to begin execution by virtue of a matching set of data objects or tokens appearing on the input data-flow pins of the widget. When they complete, they produce a set of resultant data tokens on their outputs that then become part of the environment (persistent or otherwise). Thus, a widget that is capable of processing images would specify at least one input pin of type image such that when an image passed through the intake pipe, it could appear at the widget's input pin and cause it to

execute. By contrast, conventional systems allocate execution time to a program without knowledge of what it is actually doing, and it is up to the program itself to seek out and acquire its required inputs. To do this, the program requires detailed knowledge of its environment, and the need for this knowledge reduces the generality of the program and increases the overall rigidity of the system thus making it resistive to change and more likely to develop a 'stovepipe' topology. By adopting the radical approach to attacking the problem, the present invention provides an open-ended architecture on which intelligence and similar applications can be built.

CLAIMS

1. A system for managing knowledge represented by an incoming data stream, comprising:

2. a system for converting incoming unstructured data into a well described normalized form;

3. a types system for accessing and manipulating data held either in memory or in persistent storage in its normalized binary form;

4. one or more 'widgets' within the system that can freely and effectively operate on data types they have never before encountered simply by knowledge of the 'type' of data involved as determined by the types system;

5. an 'ontology' or world model that represents and contains the items and fields necessary for the target system, wherein the ontology fully specifies the form of the normalized binary data;

6. a memory management system, tied to the ontology, wherein such system splits any incoming data into one or more portions directed to one or more data containers and which defines the structure of and access to any persistent storage containers that are required to store the data;

7. a query system, wherein such system may be used to query each container to retrieve portions of such a composite object

8. a software creation system, wherein all database tables and queries are auto-generated from the ontology;

9. a user interface (UI) to display and interact with data within the system;

10. a memory collection system that forms collections of datums, and enables manipulation and exchange of these collections both within the local machine as well as across the network; and

an automated storage system, wherein such automated storage system is capable of storing data in offline, near line, or cache based storage for automated retrieval.