COLAB: A Laboratory Environment for Studying Analyst Sensemaking and Collaboration

Clayton T. Morrison and **Paul R. Cohen**
Center for Research on Unexpected Events (CRUE)
USC Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, California 90292
{clayton,cohen}@isi.edu

Abstract

COLAB is a laboratory for studying tools that facilitate collaboration and sensemaking among groups of human analysts as they build interpretations of unfolding situations based on accruing intelligence data. The laboratory has three components. The *Hats Simulator* provides a challenging problem domain involving thousands to millions of agents engaged in individual and collective behaviors, a small portion of which are terrorist. The second component, the *AIID Bayesian blackboard*, is an instrumented working environment within which analysts collaborate to build their interpretation of the problem domain. The third component is a web-based user interface that integrates the *Trellis* hypothesis authoring and management tool with a query language to allow human analysts to interact with AIID and each other. Looking to the big picture, COLAB is not "merely" a laboratory for studying collaboration and shared knowledge creation. COLAB is a tool to explore and develop the analyst working environment of the future, in which analyst tools and methods for collaboration in edge organizations are developed and tested. We present COLAB and its components and outline our plans for the system.

Introduction

The COLAB Project brings together a large scale terrorist simulator, a collaborative intelligence analysis environment, and a user interface to produce a prototype end-to-end system for intelligence analysis. The simulator is operational and has been used in several studies, including providing data for part of the AFRL EAGLE project and assessing algorithms for relational data mining (Cohen & Morrison 2004; Morrison *et al.* 2005). The intelligence analysis environment (Sutton *et al.* 2003; 2004) and interface are under development. The complete system has three intended applications:

1. An environment for training intelligence analysts
2. A testbed for intelligence analysis tools
3. A configurable laboratory to test models of command and control organization structure in an intelligence analysis setting.

By way of introduction, we focus here on the third application. In their book *Power to the Edge* (2003), Alberts and Hayes explore a vision of the future of command and control that changes fundamentally the relationships between organization members, how they interact, the information they have access to, and their decision making capabilities. In an *edge* organization, the traditional hierarchical structure of an organization is flattened so that decision-makers are no longer insulated from information directly available to those "in the field." Alberts and Hayes propose that, properly realized, an edge organization has greater flexibility to respond to crises and changes (self-organization and self-synchronization), and decision-making is enhanced by ensuring all of the relevant information is available, rather than being "lost" in, or obscured by, the bureaucracy of middle-management. This is an exciting vision and would seem to address many of the challenges faced by the nation's security organizations. But there are many details to be worked out. How exactly do we implement an edge organization? And, are edge organizations appropriate for large-scale intelligence analysis tasks? One approach to answering these questions is by computer simulation of organization work flow and information access (Levitt *et al.* 1994; Nissen & Levitt 2004). COLAB provides a complementary approach as a configurable laboratory environment in which to conduct controlled experiments with actual human analysts working together on an artificial but challenging intelligence analysis problem.

Consider the following scenario. Several human analysts work together in the COLAB environment to develop an interpretation of events in a simulated world. Their goal is to identify and stop terrorist agent activities in the simulator while trying to keep their costs low. Obtaining information about the simulation is expensive and there are penalties for making false arrests and failing to identify terrorist plots. By design, each player has a different view of the information in the simulated world and none has all the relevant information. Each player has her own workspace where she can store and process information that she gathers from the simulator. The players collaborate via a shared workspace where they can post hypotheses and data they think is relevant to the larger analysis. This shared space becomes the collective interpretation of the state of the simulator. By monitoring this interpretation a player can identify trends, patterns, and gaps in the corporate intelligence. She will also develop trust (or mistrust) in her colleagues by noting

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JUN 2005		**00-00-2005 to 00-00-2005**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
COLAB: A Laboratory Environment for Studying Analyst Sensemaking and Collaboration	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **USC Information Sciences Institute,Center for Research on Unexpected Events (CRUE),4676 Admiralty Way Suite 1001,Marina del Rey,CA,90292**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**13**	

the quality of their analyses.

We envision COLAB being configurable along the following dimensions to test the strengths and weaknesses of proposed analysis environments:

1. Command and control structure: Participants may be organized into a strict hierarchy; analysts at the "bottom" are responsible for particular aspects of analysis, with supervisors distilling analyses and reporting to decision-makers who actually take actions (e.g., raising an alert) in the simulator. Or, each analyst may have the same access to information and there is no explicit reporting hierarchy; each analyst is equally responsible for analysis and decision-making, and any command structure must be negotiated.
2. Communication channels: Analysts may be allowed to communicate directly via text messaging or voice, or only through postings to a shared workspace. Analysts may be in the same room or in separate locations.
3. Information access: Access to information may be varied. Analysts may each have access to only one aspect of the problem domain and must collaborate to build a global picture of the unfolding scenario. Alternatively, access to domain information may overlap and it is up to the group to determine how to divide information analysis responsibilities.
4. Trust and information quality: Information quality may be varied, for example by modeling information sources. Analysts must determine what information sources are trustworthy, and likewise must learn to establish trust in their fellow analysts. As a variation on this theme, one or more analysts could be "plants" in the experiment, purposely conducting poor or inconsistent analysis. How are variations in information and analysis quality within the organization identified and managed?
5. Corporate memory and knowledge preservation: After conducting analysis for a period of time, an original analysis team member may be replaced by a new analyst. What is the effect of introducing new analysts in the middle of sensemaking? How do they get up to speed? How is prior knowledge effectively preserved and communicated?
6. Assessing cognitive load: Analysts tell us they already receive too much information. How much information should a single analyst be responsible for managing? How do we determine when analysts have reached that limit or have cycles to spare? How can analysts communicate their cognitive load status, and how do organization conditions affect this communication and the ability to shift work load?
7. Affects of disruption to sensemaking: There are many interesting case studies of how organizations respond to crises (e.g., Weick 1993). What happens if the analysis environment is stressed? We may test this by making portions of the workspace inaccessible or degrade communication channels during an analysis session. We can study how analysts maintain or recover sensemaking in the face of disruption.

In the following sections, we present the components of the COLAB system. We begin with an overview of the Hats Simulator, which provides a challenging intelligence analysis problem domain. We then present AIID, a blackboard system that serves as the core to the analysis working environment. We then turn to the web-based interface to COLAB. The interface incorporates the Trellis argument authoring tool as a tool for hypothesis representation and the query language for accessing and manipulating information in AIID. We conclude with a brief walkthrough of the system, a discussion of related work, and future plans for COLAB.

The Hats Simulator

The Hats Simulator is designed to be a light-weight proxy for many intelligence analysis problems. The simulator is implemented and currently manages the activities of up to a hundred thousand agents. The emphasis in Hats is not domain knowledge but the management of enormous numbers of hypotheses based on scant, often inaccurate information. By simplifying agents and their elementary behaviors, we de-emphasize the domain knowledge required to identify terrorist threats and emphasize covertness, complex group behaviors over time, and the frighteningly low signal to noise ratio. Playing the game successfully requires collaboration, making this domain ideal for studying analyst group sensemaking.

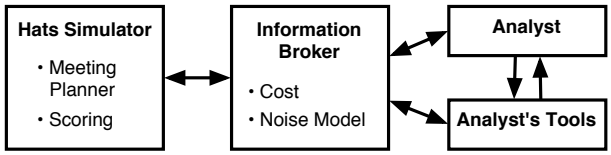

Figure 1: The Hats Simulator architecture

The Hats Simulator consists of the core simulator and an information broker (see Figure 1). The simulator core includes a generative meeting planner that plans hat behaviors based on the context of the unfolding scenario. The core also keeps track of analyst actions, assessing a score at the end of a scenario run. The information broker is responsible for handling requests for information about the state of the simulator and thus forms the interface between the simulator and the analyst and her tools. In the following sections we describe the Hats domain, a discussion of classes of hypotheses about the domain, the information broker, analyst actions, and scoring. (Details about population generation, meeting planning and the information broker can be found in Cohen & Morrison 2004, and Morrison *et al.* 2005.)

The Hats Domain

The Hats Simulator models a society in a box consisting of many very simple agents, called *hats*. Hats get its name from the classic spaghetti western, in which heroes and villains are identifiable by the colors of their hats. The Hats society also has its heroes and villains, but the challenge is to identify which color hat they should be wearing, based on

how they behave. Some hats are *known* terrorists; others are *covert* and must be identified and distinguished from the *benign* hats in the society.

Hats is staged in a two-dimensional grid on which hats move around, go to meetings and trade capabilities. The grid consists of two kinds of locations: those that have no value, and high-valued locations called *beacons* that terrorists would like to attack. All beacons have a set of attributes, or *vulnerabilities*, corresponding to the *capabilities* which hats carry. To destroy a beacon, a task force of terrorist hats must possess capabilities that match the beacon's vulnerabilities, as a key matches a lock. In general, these capabilities are not unique to terrorists, so one cannot identify terrorist hats only on the basis of the capabilities they carry.

The Hats society is structured by organizations. All hats belong to at least two organizations and some hats belong to many. Terrorist organizations host only known and covert terrorist hats. Benign organizations, on the other hand, may contain any kind of hat, including known and covert terrorists.

Hats act individually and collectively, but always planfully. In fact, the actions of hats are planned by a generative planner. Benign hats congregate at locations including beacons. Terrorist hats meet, acquire capabilities, form task forces, and attack beacons. The purpose of the planner is to construct an elaborate shell game in which capabilities are traded among hats in a potentially long sequence of meetings, culminating in a final meeting at a target. By moving capabilities among hats, the planner masks its intentions. Rather than directing half a dozen hats with just the capabilities required for a task to march purposefully up to a beacon, instead hats with required capabilities pass them on to other hats, and eventually a capable task force appears at the beacon.

The Information Broker

The information broker provides information about the state of the Hats world. The information broker will respond to questions such as *Where is Hat_{27} right now?* It will also provide information by subscription to analysts' tools, which in turn make information broker requests. For example, a tool might process requests like, *Identify everyone Hat_{27} meets in the next 100 steps*, or, *Tell me if Hat_{27} approaches a beacon with capabilities c_1, c_7 or c_{29}*. In later sections we describe facilities available in COLAB for making similar requests.

Some information is free, but information about states of the simulator that change over time is costly. Free information includes information about the population (who the known terrorists are), the simulator world (world-map dimensions), and some event bookkeeping (locations of attacks, a list of currently arrested hats).

Other types of information require payment and the more one pays, the more accurate the report returned. The relation between payment and noise and how requested reports are made noisy is described in Cohen & Morrison (2004) and Morrison *et al.* (2005). There are five elementary report types an analyst can pay for: (1) the hats at a specified location in the Hats world, (2) the location of a specific hat, (3) the capabilities currently carried by a hat, (4) the list of participants in a suspected meeting (the analyst must provide the time and location of the meeting), and (5) information about capability trades that occurred in a meeting.

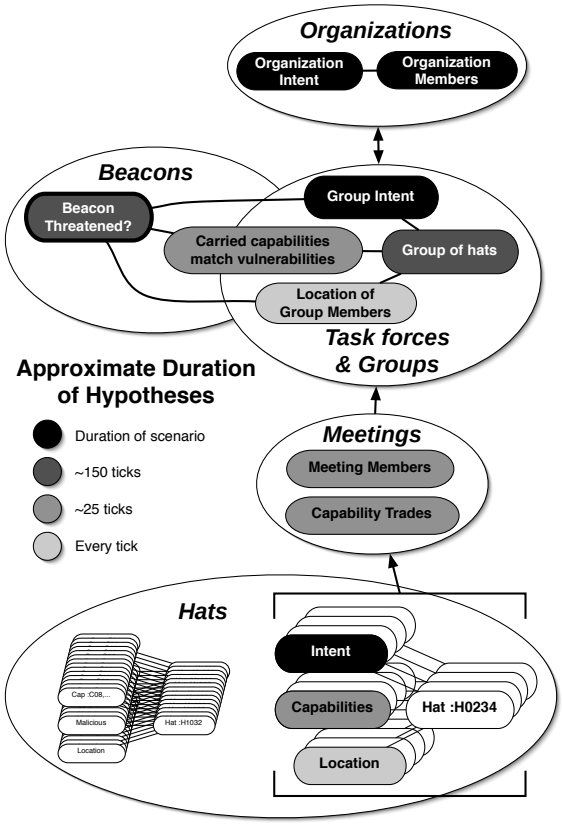

Figure 2: Schematic representation of classes of hypotheses about the Hats domain. Large ovals indicate hypothesis classes, small ovals represent specific hypothesis types. The shades of gray indicate estimated duration of hypothesis accuracy or relevancy. Links between ovals indicate dependency relations between hypothesis types. The `Beacon Threatened?` hypothesis type is outlined in bold, indicating that it is one of the most important target hypotheses of analysis.

The Space of Hats Hypotheses

Analysts rely on reports returned by the information broker to construct a model of observed events that can explain and predict hat behaviors. The goal of this analysis is to identify threats to beacons and, if possible, arrest terrorist task force hats before they carry out an attack. In the process of doing so, analysts will keep track of hat locations, the capabilities hats carry, and the meetings they participate in. They will also construct hypotheses about the intents of hats, associate hats that appear to be members of a task force, and eventually form hypotheses about organization membership and intent.

Figure 2 represents two properties of Hats domain hy-

potheses that analysts will need to express:

1. Dependencies of hypotheses on one another are represented by the links and arrows in the figure. For example, a hypothesis that there exists a task force of hats depends on an observed pattern of meetings between hats; hats that are members of a task force are, in turn, likely members of the same organization. Similarly, hypotheses about the intents of individual hats inform hypotheses about the intents of the task forces and organizations to which they belong.
2. Properties of the simulation change at different time scales, so hypotheses about aspects of the domain will have to be updated at different rates. Figure 2 represents four different relative time scales using different shades of gray. For example, the location of a hat changes almost every tick, the set of capabilities a hat is carrying updates roughly every 25 ticks, but the intent of a hat (whether it is terrorist or benign) remains constant throughout the game.

In the later section describing the COLAB interface, we describe methods for representing the relations between hypotheses as well as representing time.

Actions

We may not be able to stop an attack, but if we know it is coming, we can prepare and minimize loss. This is the inspiration behind modeling *alerts*. Each beacon can be in one of three alert levels: off (default), low or high. These correspond to the conditions of no threat, a chance of an attack, and attack likely. The analyst decides which level a beacon alert is set to, but the Hats Simulator keeps track of alert states over time and whether an actual attack occurs while the state is elevated. The simulator keeps statistics including counts of hits (occurrences of attacks during elevated alerts) and false positives (elevated alerts that begin and end with no beacon attack occurring). The goal of the analyst is to minimize the time beacon alerts are elevated. High alerts are more costly than low ones. On the other hand, if an attack does occur on a beacon, a high alert is better than a low alert, and a low alert is better than none.

Analysts can also issue *arrest warrants* for hats in order to prevent beacon attacks. Arrests are successful only when the targeted hat is currently a member of a terrorist task force. Attempted arrests under any other conditions, including hats that are terrorists but not currently part of a terrorist task force, result in a false arrest (a false positive). Under this model, a hat can be a terrorist but not be guilty of any crime. Unless terrorist hats are engaged in ongoing terrorist activities, their arrest incurs penalties. While this is a simple model, it places realistic constraints on the analyst's choice of actions. Furthermore, successful arrests do not guarantee saving beacons. A beacon is only attacked when some subset of members from a terrorist task force successfully carry the capabilities matching the target beacon's vulnerabilities to a final meeting at that beacon. It is possible to successfully arrest a terrorist task force member but the other terrorist taskforce members still have the capabilities required to attack the beacon. However, if the analyst successfully arrests a terrorist task force member carrying required capabilities that no other task force member has, then the final meeting of the task force will take place but it will not be attacked. This is counted as a beacon save.

Scoring

As the simulation progresses, three kinds of costs are accrued:

1. The cost of acquiring and processing information about a hat. This is the "government in the bedroom" or intrusiveness cost.
2. The cost of falsely arresting benign hats.
3. The cost of harm done by terrorists.

The skill of analysts and the value of analysis tools can be measured in terms of these costs, and they are assessed automatically by the Hats Simulator as analysts play the Hats game. At the end of a simulation run, a final report is generated that includes the following four categories of scores:

1. Costs: the total amount of "algorithmic dollars" spent on information from the Information Broker.
2. Beacon Attacks: including the total number of terrorist attacks that succeeded and the total number of attacks that were stopped by successful arrests
3. Arrests: the number of successful arrests and the number of false arrests (false positives)
4. Beacon Alerts: the number of low and high alert hits and false positives.

AIID

The second component of COLAB consists of AIID: an **A**rchitecture for the **I**nterpretation of **I**ntelligence **D**ata. AIID is based on a blackboard architecture, and is designed to represent relational data and integrate a variety of intelligence analysis algorithms as problem solving components. In work going on in parallel to COLAB development we are developing AIID as a *Bayesian blackboard* system, combining the technologies of blackboard systems and incremental construction of Bayesian belief networks from network fragments (Sutton *et al.* 2003; 2004). Our goal is to eventually include Bayesian network construction in COLAB, but here we focus on the components of AIID used to implement COLAB analysis workspaces. We first introduce blackboard systems and then describe the COLAB blackboard components.

Blackboard Systems

Blackboard systems are knowledge-based problem solving environments that work through the collaboration of independent reasoning modules (Engelmore & Morgan 1988; Nii 1989; Corkill 1991). More recently blackboards have been recognized as platforms for data fusion (Corkill 2003). They were developed in the 1970s and originally applied to signal-processing tasks. The first, HEARSAY-II (Erman *et al.* 1980), was used for speech recognition, employing acoustic, lexical, syntactic, and semantic knowledge. Other

systems were applied to problems as diverse as interpretation of sonar data, protein folding, and robot control (Nii 1989).

Blackboard systems have three main components: the blackboard itself, knowledge sources, and control. The *blackboard* is a global data structure that contains hypotheses or partial solutions to the problem. The blackboard is typically organized into *spaces* representing levels of abstraction of the problem domain. For example, HEARSAY-II had different levels for phrases, words, syllables, and so forth. *Knowledge sources* (KSs) are small programs which post results of local computations to the blackboard. (Ideally, knowledge sources interact only by observing and posting to the blackboard.) Different KSs use different types of knowledge: for example, one might use a grammar to generate words which are likely to occur next, while another might detect phonemes directly from the acoustic signal. While no single knowledge source can solve the problem, working together they can. Getting knowledge sources to "work together" is the task of blackboard *control* (Carver & Lesser 1994). Generally it works like this: KSs watch for particular kinds of results on the blackboard; for instance, a phrasal KS might look for hypotheses about adjacent words. When a KS is "triggered" it creates a *knowledge source activation record* (KSAR) in which it requests the opportunity to run, make inferences, and modify the blackboard. These KSARs are ranked, and the top-ranked KSAR is invited to do its work. Just as knowledge sources are used for manipulating data on the blackboard, the control framework of the blackboard may also be broken up into control knowledge sources, each representing knowledge about aspects of the control problem. Domain and control KSs are often distinguished from one another because of the roles they play in the functioning blackboard.

The operation of a blackboard system can be seen as a search for hypotheses that explain the data at each level of abstraction, using the KSs as operators. Rather than search bottom-up (i.e., from the data level to the most abstract level) or top-down, blackboard systems can search opportunistically, dynamically rating KSARs based on the current data and on the partial solutions that exist so far.

The COLAB Blackboard

Blackboard Spaces Figure 3 depicts the COLAB blackboard and its component spaces. The lower three spaces are responsible for representing reports from the information broker as atomic assertions about entities and relations in the Hats domain. Take, for example, an information broker request about the members of a meeting hypothesized to have taken place at a particular location and time. The report returned from the information broker is posted to the *Raw Report* space and consists of a list of hats that were likely at the meeting. There are several entities and atomic relations this report represents: that there are hats, there was a meeting, the meeting took place at that time and location, at that time the hats were at that location, and the hats were participants in the meeting. We want to decompose the report into these atomic assertions because we want the analyst to be able to refer to each component individually or manipulate

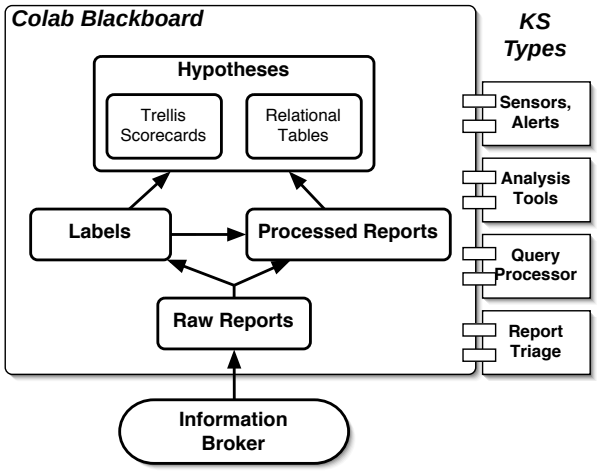

Figure 3: The COLAB blackboard. Each box on the blackboard represents a blackboard space; the *Hypothesis* space contains two subspaces. The oval at the bottom of the figure represents the information broker and the tabs to the right represent types of KSs. Arrows indicate flow of information that results from knowledge source processing.

and combine the components in novel ways. For example, the analyst may want to ask how many meetings a given hat was in over a period of time; or the analyst may only want to know where that hat was at that time, irrespective of what meeting the hat was in. Later we will describe the query language that makes this possible.

We want this decomposition, but we also want to preserve the information that each atom was purchased from an information broker request. Recall that most information from the information broker comes at a price, and the level of payment determines the accuracy of the report returned. Each atomic assertion is therefore stored on the *Processed Reports* space along with (1) the time at which the information broker request was made, (2) the time at which the event was reported as having taken place in the Hats world, and (3) the level of payment. Payment level serves as a proxy for how reliable the report is.

Finally, each atomic relation and entity is represented by a "label" in the *Labels* space. Labels are linked to each report that references them. For example, if Hat_{27} took place in the meeting from the example above, then there is a label for Hat_{27} that is linked to the atomic report asserting that it participated in the meeting and also the report that it was at that location at that time. (In the same way, there is a label identifying the meeting and it is linked to reports about the meeting.) If a referenced entity or relation does not already exist, a new label is created for it when the report is processed. Labels also have other sources than reports; as we will discuss below, labels comprise the vocabulary available to the analyst to express hypotheses. Labels also serve as an index into the space of processed reports and are used both in query processing and as an index for blackboard browsing.

The *Hypotheses* space of the COLAB blackboard is reserved for representing analyst work. The space hosts two subspaces, one for representing data structures used by Trellis, the other for analyst-defined hypotheses and the results of queries. We will discuss these further in the next section.

Knowledge Sources A class of KSs called *report triage* KSs handle processing information broker reports and updating the Labels and Processed Reports spaces. Another class of KSs consist of algorithms available to the analyst as analysis tools or "services." The beauty of the blackboard architecture is that it is specifically designed to facilitate collaborative software (Corkill 2003); as long as these algorithms can read representations on the blackboard and write output to the blackboard that is interpretable by other algorithms, they can participate in blackboard processing. Some examples of analysis services include algorithms for assigning suspicion scores to hats (Galstyan & Cohen 2005; Macskassy & Provost 2002), identifying community structure (Adibi, Cohen, & Morrison 2004; Newman 2003), and reasoning about behaviors over time. Some services may require their own blackboard spaces for specialized processing, such as a graph representation used for community finding, but the results of any service are reported to the Raw Reports space as a report, just like a report from the information broker.

We are currently prepared to offer an implementation of Newman's (2003) community finding algorithm as a service and plan to add more services in the future; service configurations will also depend on the role of the laboratory in experimental design. The analyst will be responsible for running or scheduling any services as well as specifying what data on the blackboard they take as input.

There are two other classes of KSs whose functionality will be discussed in more detail in the next section. One set of KSs handles processing of queries issued by users or originating from other knowledge sources. The other set includes user-defined knowledge sources such as *sensors* and *alerts*. Treating sensors and alerts as knowledge sources allows them to be scheduled for repeated activation and run in the background, automating routine checks of conditions such as whether a hat on a watchlist has moved within some distance of a beacon. Sensors and alerts can themselves issue queries.

Control Blackboard control in the first release of COLAB will be basic, consisting of a priory queue-based agenda shell. Report and query processing will have high priority, pushing information onto the blackboard for use as soon as reports arrive and queries are made. Users will then be able to assign priorities to sensors, alerts and other available analysis algorithms. One approach to blackboard control we are investigating includes assessing the value of the information we may request from the information broker. In some cases, the utility of certain information is overshadowed by its cost. Value of information is a kind of control knowledge and would be embodied in a control knowledge source. In general, we treat control as an open development issue in COLAB that will be shaped by performance issues expected to arise once the system is operational with multiple users.

Finally, COLAB will support multiple collaborating analysts. Each analyst will have their own configuration as depicted in Figure 3. They will ask for information from an information broker and have their own local store of collected reports. There will also be a *shared* Hypotheses space to which they can publish portions of their own blackboard spaces to make their analyses available to others.

Labels and Representing Hypothesis

Analysts need to be able to express their hypotheses and reasoning about evidence collected in the report spaces. At the same time, we want these expressions to be in a form that permits processing by knowledge sources so that, for example, an analyst's expressions can be the subject of a query or other KS processing. Ideally, the analyst would write free text and natural language processing techniques would automatically translate the text into a formal expression that can be used by any component of the system.[1] We take a middle path that mixes free text with formal vocabulary. Fortunately, the ontology of the Hats domain is so simple that we can go far with a limited vocabulary. In COLAB, the contents of the Labels space provide that vocabulary.

From processed information broker reports we get labels for hats, capabilities, beacons, and meetings. We also get some simple atomic relations, such as "has-capabilities." However there are other aspects of the hats world that the analyst will need to represent. A small set of additional label types are provided. Some examples include: *beacon-threatened*, *group*, *organization*, *overlap*, *is-malicious* and *is-benign*. The first three types represent specific event and entity types. Instances of these types are created by appending a unique number to the the label name. For example, `beacon-threatened-15` represents a specific beacon-threatened event, and `organization-05` is a particular hypothesized organization. The group label is used both to represent hypothesized task forces as well as more general groupings of hats. *Overlap* expresses a binary relation and can be used to express that one group overlaps another in membership, or that capabilities carried by a group overlap with the vulnerabilities of a beacon. The last two label types are unary predicates. For example, combining `organization-05` with `is-benign` asserts that organization 05 is benign. All of these labels are linked to any instances of their use.

Analysis tools may also generate instances of these (and other) label types as a result of their operation. For example, a group finder algorithm will return a set of hypothesized groups consisting of lists of hats. These results are posted to the Raw Reports space and processed, resulting in group labels that link to the corresponding reports on the Processed Reports space.

[1] In the section on related work we discuss some promising methods that are beginning to bridge the gap between free text and formal representation. We will explore incorporating some of these techniques in future COLAB development.

The COLAB Interface

Up to this point we have described the problem domain and the core architecture supporting the analyst working environment. We now describe the third component of COLAB: the user interface.

The first decision we had to make was what platform to implement the interface in. Making it native to our development platform[2] would enable tight integration with the COLAB blackboard and provide high performance, interactive graphics. However, we want the laboratory to be easily deployed in a variety of different conditions depending as little as possible on specific hardware and software. For this reason, we decided to make the interface web-based, running in a standard web browser. In this configuration, AIID and the Hats Simulator run on their own server and anyone with a computer connected to the internet and running a modern web browser will be able to connect to COLAB and participate as an analyst. This also facilitates future multi-analyst participation.

The central theme of COLAB interface design is *information management*. This means making information stored on the blackboard as accessible as possible and providing mechanisms for analysts to author and manage their hypotheses. Information management is an open research challenge that includes issues in knowledge engineering, query languages, data mining and data visualization. Our design philosophy is to start simple and use existing technologies as much as possible. For hypothesis representation, we are integrating the Trellis argument authoring tool. For information access we are implementing a relational query language similar to SQL. And our initial interface for browsing the blackboard will be hypertext-based.

Trellis

In the discussion about Hats domain hypotheses and their relations (see Figure 2) we noted that analysts will need to express how hypotheses depend on one another. For example, the hypothesis that a beacon is threatened (is about to be attacked) depends on lower level hypotheses about the members of a group, whether the group has the capabilities to attack the beacon, and the group's intent. Group intent, in turn, depends on the intent of the individuals in the group. Analysts need a way to make these relationships explicit so that they can express evidential support and frame the structure of the argument supporting (or denying) a target hypothesis. The Trellis argument authoring tool makes this possible.

Trellis was originally developed as an interactive tool to assist users in constructing arguments based on sources gathered from the internet (Gil & Ratnakar 2002). Users create statements that combine free text and references to internet documents. These statements are then combined and related using a set of structured argument connectors. In the original version of the system, a variety of connectors were provided, such as "is elaborated by," "is supported by," and "stands though contradicted by." This original system, now dubbed "Rich Trellis," was found to offer the user too many options, leading to multiple ways of expressing the same line of reasoning and inconsistent use of the argument connectors (Chklovski, Ratnakar, & Gil 2005). Both of these sources of ambiguity are problematic for managing and communicating hypotheses. Two new versions of Trellis have been developed to ameliorate these problems. One of these, *Tree Trellis* is a "lite" version of Rich Trellis that restricts the user to two kinds of hierarchical argument connectors: pro (supporting) and con (against). Preliminary data suggest that Tree Trellis users use argument connectors more consistently and arguments are subsequently easier to compare (Chklovski, Ratnakar, & Gil 2005).

We have chosen to adopt Tree Trellis as the COLAB hypothesis authoring tool because it expresses the basic hierarchical support/dissent relationships between statements that we believe analysts will need. Tree Trellis also provides a well-designed, intuitive user interface. The left half of the browser window in Figure 4 shows an example of the Tree Trellis interface. The interface currently displays an argument supporting the beacon-threatened hypothesis. The top-level statement (the "target hypothesis") asserts that `group-01` threatens beacon `B012`. This claim is supported by two additional statements, that `group-01` has capabilities that overlap the vulnerabilities of the beacon and that `Group 1` is malicious. Capability overlap is in turn supported by evidence that the capabilities are carried by individual hats in the group. The example also includes a "Con" relationship, expressing a statement against the group having the required overlapping capabilities: hat `H0540` is no longer carrying the remaining required capability `C14`. Report 1719 is cited as evidence for this claim.

We are currently working on integrating Tree Trellis with the COLAB blackboard. Part of this integration includes making it easy to reference COLAB blackboard labels in statements. Labels are depicted in Figure 4 as names that begin with a colon. Under the hood, Trellis arguments are represented as "score cards" that are communicated between the Trellis engine and the *Trellis Scorecards* blackboard space. Because scorecards are full representations of the tree argument and include blackboard label references, they can be the subject of queries and other knowledge source processing. For example, if the analyst asks for all arguments that include hats `H0328` and H1024, the argument in our example will be among those returned.

In our discussion of Figure 2 we also noted that the hypotheses depend on the relative rate at which properties of the Hats world change. In COLAB, each hypothesis is treated as a *fluent* (McCarthy 1963): a proposition whose truth value is extended over time. We represent this property in Figure 4 by the two numbers following the word "Tick." The number in parentheses represents the time the hypothesis was first asserted while the number outside represents the last time the hypothesis was updated and asserted as being true. For example, the statement that hat `H1024` has capabilities `C02` and `C03` was first asserted in tick 52 and still holds at tick 73 (which in this example happens to be the current tick). The statement that hat `H0540` has capability `C14`, however, has not been updated and is considered to not hold after tick 72. One of the challenges of analysis is determining the temporal boundaries of hypotheses.

[2] Macintosh OS X 10.2.8 running Macintosh Common Lisp 5.1

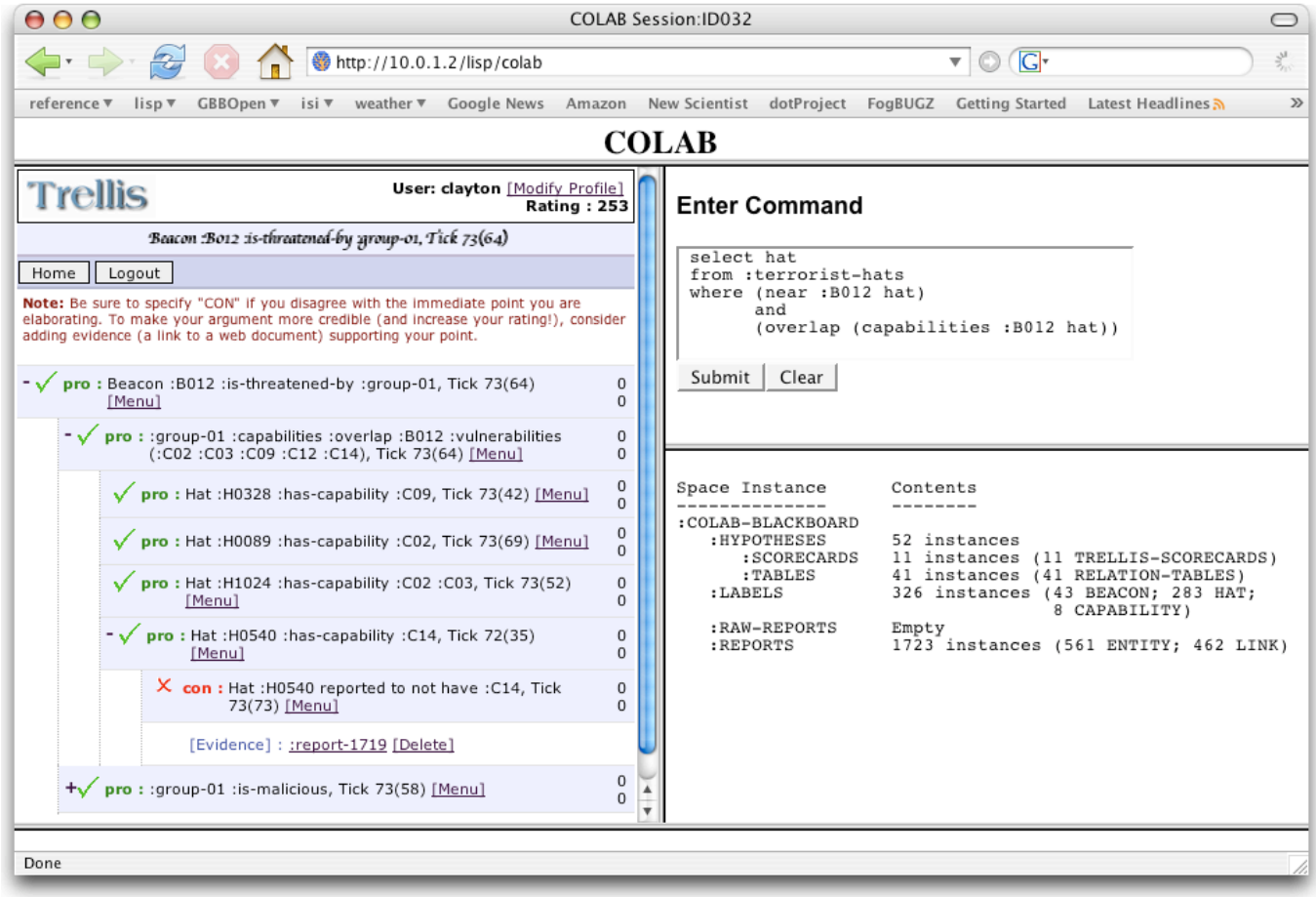

Figure 4: The COLAB web interface. The left half of the browser window contains the interface to the Trellis argument authoring tool. The upper right of the browser window contains the field for entering commands and queries. In the bottom bottom right a view of the top-level of the blackboard is displayed.

Query Language

During an analysis session, the blackboard will rapidly fill with many reports about events in the Hats world, the results of analysis tools and analyst-authored hypotheses (represented as Trellis score cards). In order to manage and manipulate this data, we are implementing a query language facility based on the relational database model. The language syntax is based on the SQL, a well-studied language that is intuitive and has a long, successful history in practical database applications (Elmasri & Navathe 1999).

In the relational model, an individual fact is represented as a set of attributes with values. Facts are collected in tables in which each row represents a fact and each column corresponds to an attribute; an attribute value corresponding to the attribute of a particular fact is stored in the cell indexed by the attribute column and fact row. Queries specify information to extract from tables, and the results of queries are expressed in new tables. We are setting up the query framework in COLAB so that the spaces in the blackboard are treated as a tables, and all queries result in tables stored on the (relational) Table space of the blackboard.

```
select trade, giver, taker, capability,
       meeting, tick
from reports
where (range ticks 10)
```

Figure 5: Example query retrieving all trade reports in the past 10 ticks

The upper-right frame of the browser window in Figure 4 shows the command entry field and contains an example query. This query specifies that a table of hats should be retrieved from the :terrorist-hats table, where each hat that appears in the new table must be near beacon `B012` and have capabilities that overlap with the beacon's vulnerabilities. The table resulting from this query will be displayed in the lower-right of Figure 4 or in a separate window.

Because all reports are indexed by time, we are also providing facilities for specifying individual time intervals. Figure 5 is an example of a query asking for all reports of capability trades in the past 10 ticks. The query creates a new

table, where each fact is about a reported trade and includes the hat that gave the capability, the that received the capability, the label of the meeting in which the trade took place, and the time when the trade took place. If no specific time or interval is specified, the query is assumed to be asking about the reports in the current tick.

By default, a table resulting from a query remains in the state it was in when the query was executed. Tables can also be specified to update dynamically. A dynamic query table looks like any other except that every time the table is viewed, the query is run again and the table is updated to contain the most up-to-date information.

Each table stored on the Tables space is indexed by a unique label on the Labels space. These labels can be used to reference tables in hypothesis arguments and other queries.

Knowledge Source Toolkit, Sensors and Alerts

Knowledge sources provide another facility for analyst information management. We are developing a *knowledge source toolkit* as a set of KS templates that analysts can specialize for user-defined tasks. One such task is scheduling repeated information broker requests and other blackboard queries (in fact, all queries are processed by query KSs). An analyst selects a query scheduling KS and fills in its slots, including the query to be executed[3], how often it is executed, and what to do with the query results.

Another template allows analysts to define *sensors*. In addition to specifying a query and a schedule for execution, the analyst also defines a condition (using the same language used to specify conditions in queries) that the sensor checks each time it is executed. Based on the results of the condition test, the KS can be directed to perform simple actions, including triggering activation of other KSs, or updating a dynamic query table.

Alerts are a special class of sensors whose actions include sending messages to be displayed on the analyst user interface (e.g., the bottom-right pane of the browser window in Figure 4).

The following are examples of KSs built out of the toolbox templates:

Meeting Alert Any time two or more hats from a specified set of hats (e.g., all hats whose locations have just been reported) meet at the same location for more than two ticks, send an alert to the user that a meeting may have just taken place (Along with location and time). Another sensor KS may be defined to trigger when the meeting alert is issued; this sensor may then send query to the information broker asking whether a meeting has taken place at that location.

Watchlist Scheduled Query This KS updates a "watchlist" dynamic table to include any hats whose number of meetings with known terrorists is above some threshold. Alternatively, the KS may schedule execution of a suspicion scoring analysis service and the suspicion scores of hats above some threshold are included in the table

[3] if it is an information broker request, the analyst also specifies how much to pay for the request.

Beacon Vulnerability Sensor After each update to the watchlist above, check whether any hats on the watchlist have capabilities that overlap a specified beacon's vulnerabilities. If so, trigger a beacon threat alert KS.

Beacon Threat Alert Triggered by the Beacon Vulnerability Sensor, this KS tests whether hat(s) triggering the vulnerability sensor are within some distance of the beacon. If so, then send an alert to the analyst.

The Knowledge Source Toolkit takes steps toward a language for defining special purpose knowledge source, blurring the line between human control and blackboard automation.

Blackboard Browsing

The Trellis hypothesis authoring tool and the query language allow the analyst to visualize relations between hypotheses and other information stored on the blackboard. Still, the analyst will likely want a more generic method for browsing blackboard contents, including the contents of spaces and a representation of currently knowledge sources and their state of activation. We are developing a hyperlinked navigation tool for intuitive blackboard browsing. In the lower right space of Figure 4 is simple textual summary of the blackboard spaces and their contents. In the browser, the names for spaces will be hyperlinks, which when selected will display the specific contents of the space. These contents will also be represented as hyperlinked names, which when selected provide summary descriptions of the object with possible links to other objects. In future work we will explore other approaches to visualizing blackboard spaces and hypothesis relations.

Putting the pieces together

Now that we have presented each of the COLAB components and along the way discussed how they related, we can step back and again consider the system as a whole. Figure 6 is a schematic of the COLAB architecture. The schematic emphasizes the multi-agent configuration of COLAB by showing two analyst environments and their relation to the Hats Simulator and shared blackboard workspace. In this configuration, each analyst has their own information broker, which may represent and deliver reports about different aspects of the Hats domain. Both analysts interact with the information broker and COLAB blackboard workspaces through the COLAB user interface. Each analyst has their own blackboard workspace and a set of domain and control knowledge sources, including their own knowledge sources built from the knowledge source toolkit. In this configuration, analysts collaborate with one another through the shared workspace.

Implementation

COLAB is being developed in Macintosh Common Lisp (MCL) 5.1 running on Macintosh OS X. For blackboard development we are using the GBBopen blackboard framework (http://gbbopen.org/). The web interface server is written in Common Lisp and communicates with the Apache webserver (http://httpd.apache.org/) through the

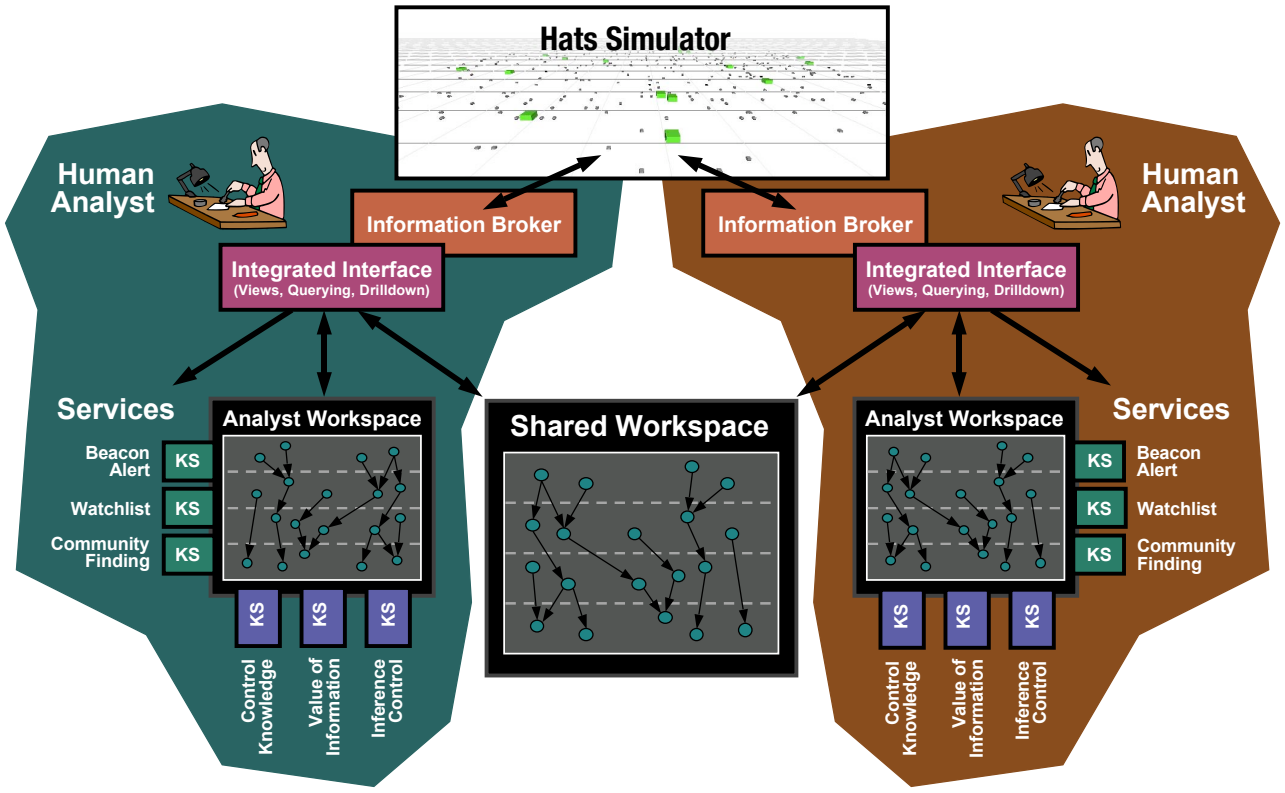

Figure 6: The COLAB architecture configured for two analyst environments.

mod_lisp (http://www.fractalconcept.com/) Apache module. We also make use of a number of available Lisp packages (http://lisp.t2100cdt.kippona.net/lispy/home). Except for MCL and Mac OS X, all of these software packages are open source.

Related Work

The COLAB project makes contact with a number of technologies and areas of active research. We restrict the following discussion to a few representative projects in each area. We have grouped the descriptions according to four topics.

Architectures

Artificial intelligence research has produced a very large family of architectures for problem solving and reasoning. Here we compare blackboards to two popular and well-studied architectures: production rule systems and case-based reasoning (CBR) systems. Blackboards, production rules and CBR systems share the same intellectual roots, so it is no surprise that much of their functionality overlaps.

In the classical conception of a production rule system, knowledge consists of if-then style rules and assertions. Similar to blackboard knowledge sources, each production rule has an antecedent and a consequent. If the antecedent is satisfied by existing assertions in a database, then it may be selected to execute the consequent of the rule. Production rules differ from deduction rules in that the rule consequent may not only add new assertions to the database, but may also delete assertions and have side-effects (such as executing other programs). An example of a long-running project based on a production rule architecture is ACT-R (Anderson & Lebiere 1998).

CBR systems take a different approach to representing knowledge and its use in problem solving. In a CBR system. previous problem solving solutions or strategies are stored as "cases." When a new problem situation is encountered, similar, previously successful case scenarios are retrieved. An attempt is made to fit one (or more) of these case solutions to the current situation. In most cases, it won't fit exactly, so the case is modified, and if successful the modification is stored as a new case for future use. Cases could be viewed as knowledge sources and their retrieval and execution is similar to knowledge source activation and selection. CBR research provides sophisticated methods for retrieval based on analogy and other mechanisms, as well as the possibility of learning (adding new cases) over time (Kolodner 1993; Watson & Marir 1994).

Capturing Knowledge and Reasoning

COLAB also makes contact with approaches to capturing, representing and communicating human knowledge and reasoning. The general goal is to bridge the gap between intuitive human forms of communication, such as iconic or natural language expressions of human knowledge, and lan-

guages with unambiguous syntax and semantics that can be processed by computer programs.

Blythe *et al.* (2001) describe a system that incorporates a number of individual knowledge acquisition tools to produce an end-to-end tool that enables non-programmers to author formal rules in the EXPECT representation and reasoning system. One of the themes of this system is interaction between the user the various tools to incrementally formalize knowledge, such as the user's preferences in planning travel. Trellis also fits into this theme. Chklovski, Ratnakar, & Gil (2005) describe the Trellis tools as exploring the tradeoffs in designing semi-formal representations so that the system can provide useful assistance while minimizing the user's effort in formalizing knowledge about the task at hand.

Trellis and several other tools are specifically designed to help users connect statements and author arguments that can be visualized and reasoned over. SEAS, the Structured Evidential Argumentation System, enables users to enter structured arguments that can then be revised in the light of new evidence (Lowrance, Harrison, & Rodriguez 2001). Similar to Trellis, analysis is captured in a tree of relevant issues and sub-issues that are assessed a score by an evidential rating system. Researchers working on the semantic web also share the vision of providing a tool that can be used link concepts and sources from digital libraries in a networked representational environment. Shum *et al.* (2003) present work on a system for representing scholarly discourse in this way. These kinds of tools are also being used to help teach formal and information reasoning. An example is the Reason!Able argument mapping tool (van Gelder 2002).

Relational Data Management and Analysis

Intelligence professionals have to deal with an overwhelming amount of information. Much of this information is becoming increasingly available in relational databases. New systems are being designed to represent, manage and analyze this data.

The Link Analysis Workbench (LAW) is designed to help a user build and refine relational patterns used to search large relational databases (Wolverton *et al.* 2003). Robust search based on pattern matching must deal with missing information. Additionally, the analyst often does not know ahead of time what they are looking for. For these reasons, LAW employs a graph-based query facility that handles approximate matches based on graph edit distance, and query results are ordered by degree of match. LAW also provides a web-based graphical user interface for visualizing of relational data as graphs and and for pattern authoring. The end result is a system in which the analyst can iteratively pose and refine graph-pattern queries to search for information

Another system designed to facilitate iterative search and refinement of data models is PROXIMITY (Neville & Jensen 2002). PROXIMITY consists of a graph database and a set of statistical relational modeling tools. PROXIMITY also includes a graphical query language called QGRAPH (Blau, Immerman, & Jensen 2002). QGRAPH is similar to the LAW graph-query facility, but also has a clear formal semantics and has been shown to have the expressive power of a subset of first-order logic with counting quantifiers.

Another feature of intelligence data is that is that it crucially involves time. This raises a host of new challenges for database management and data access. Recent work in database research is devoted to representing temporal data *streams*. The Stanford STREAM Database project is developing a data stream management system capable of handling multiple continuous queries over data streams and stored relational data (Babcock *et al.* 2002; Arasu, Babu, & Widom 2005; Arasu *et al.* 2005).

Collaboration Environments

Several of the systems already mentioned, in particular work on the semantic web and volunteer knowledge capture, could also be considered as examples of collaboration environments. Another good example of a collaborative environment is the Collaborative Virtual Workspace (CVW) (Spellman *et al.* 1997; Maybury 2001). The CVW system allows for people in different locations to interact with documents and each other in a virtual space. CVW integrates tools for audio and video conferencing, document management, chat, and graphical sketching (whiteboarding). The system has been successfully fielded in the Air Force's Joint Expeditionary Force Experiment (JEFX) and another government organization supporting 4000 active users around the globe (Maybury 2001). More recently, commercial systems have been offered that also support much of the same functionality, such as Microsoft Office Live Meeting.

Concluding Remarks

In the introduction we argued that there is a need for a configurable laboratory environment in which variables hypothesized to influence collaboration and shared sensemaking can be manipulated and studied. We have made design decisions with the goal that COLAB can be adapted to different work environment conditions. Laboratories are not a replacement for studies of professional analysts in real intelligence analysis situations. But purpose of COLAB is to provide a unique opportunity to study, in detail, analyst performance in an environment in which we have control over both the problem situation faced by analysts and the methods analysts have available for performing their analysis. Because we know the state of the Hats Simulator, we can keep track of the steps analysts take in their analysis and compare them to information that was actually available in the Hats simulation and on the blackboard. We can identify when information was available but not used and analyze situations in which communication was not effective and sensemaking floundered.

We also noted in the introduction that COLAB is intended for two other applications: a test bed for proposed analysis tools in the analyst working environment of the future, and an environment for training analysts. In the description of the COLAB blackboard we mentioned some proposed analysis tools, such as algorithms for suspicion scoring and finding community structure. In general, these tools are only useful to the extent that they help analysts do their job better. COLAB, along with the Trellis project, the Link Analysis Workbench, Proximity, and SEAS, is part of an important

class of systems that explore the space of analysis environments that bring human users into the analysis loop.

Here is where the project stands. The Hats Simulator and information broker are currently implemented. The foundation for the COLAB blackboard is implemented and reports from the information broker are automatically processed and the Labels space is updated. We have designed and are currently implementing the query language facilities. The foundation for basic web service is in place and we are working on integrating Tree Trellis. The next milestone is working prototype of COLAB for a single user, which we anticipate meeting by early summer. We will then begin user testing and extend the framework for multiple users.

Acknowledgments

Work on this project is supported by the Office of the Assistant Secretary of Defense for Networks and Information Integration (OASD/NII), through its Command & Control Research Program (CCRP), USC subcontract CCRP-COLAB 53-4540-7723. We are indebted to Dr. David S. Alberts, Dr. Mark Nissen and the Naval Postgraduate School's Center for Edge Power for leading and coordinating the Edge Power Projects. We thank Dr. Tim Chklovski for discussions and help with integrating Trellis, and Gary W. King for help with GBBopen and Ijara. Thank you also to Andrew Hannon and Zeeshan Maqbool for help with the web interface. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

References

Adibi, J.; Cohen, P. R.; and Morrison, C. T. 2004. Measuring confidence intervals in link discovery: a bootstrap approach. In *Proceedings of the ACM Special Interest Group on Knowledge Discovery and Data Mining (ACM-SIGKDD-04)*.

Alberts, D. S., and Hayes, R. E. 2003. *Power to the Edge: Command and Control in the Information Age*. CCRP Publications Series. http://www.dodccrp.org/Publications/pdf/poweredge.pdf.

Anderson, J. R., and Lebiere, C. 1998. *The Atomic Components of Thought*. Mahwah, NJ: Erlbaum.

Arasu, A.; Babu, S.; and Widom, J. 2005. The cql continuous query language: Semantic foundations and query execution. *To appear in the International Journal on Very Large Data Bases (VLDB Journal)* 14. Available as technical report at http://dbpubs.stanford.edu/pub/2003-67.

Arasu, A.; Babcock, B.; Babu, S.; Cieslewicz, J.; Datar, M.; Ito, K.; Motwani, R.; Srivastava, U.; and Widom, J. 2005. Stream: The stanford data stream management system. In *Data-Stream Management: Processing High-Speed Data Streams*. Springer-Verlag. Available as technical report from http://dbpubs.stanford.edu/pub/2004-20.

Babcock, B.; Babu, S.; Datar, M.; Motwani, R.; and Widom, J. 2002. Models and issues in data stream systems. In *Proceedings of the 2002 ACM Symposium on Principles of Database Systems*, 1–16. Available as technical report from http://dbpubs.stanford.edu:8090/pub/2002-19.

Blau, H.; Immerman, N.; and Jensen, D. 2002. A visual language for querying and updating graphs. Technical Report 2002-037, University of Massachusetts Amherst Computer Science Technical Report. http://kdl.cs.umass.edu/people/jensen/papers/tr02.html.

Blythe, J.; Kim, J.; Ramachandran, S.; and Gil, Y. 2001. An integrated environment for knowledge acquisition. In *Proceedings of the 2001 International Conference on Intelligent User Interfaces (IUI-2001)*. http://www.isi.edu/expect/papers/blythe-kim-ramagil-iui01.pdf.

Carver, N., and Lesser, V. 1994. The evolution of blackboard control architectures. *Expert Systems with Applications–Special Issue on the Blackboard Paradigm and Its Applications* 7(1):1–30. Available as a University of Massachusetts technical report: http://mas.cs.umass.edu/paper/89.

Chklovski, T.; Ratnakar, V.; and Gil, Y. 2005. User interfaces with semi-formal representations: a study of designing argumentation structures. In *Under Review for the Intelligent User Interfaces Conference 2005*. http://www.isi.edu/∼timc/papers/trellis-iui05-chklovski.pdf.

Cohen, P. R., and Morrison, C. T. 2004. The hats simulator. In *Proceedings of the 2004 Winter Simulation Conference*. http://eksl.cs.umass.edu/papers/cohen-wsc04.pdf.

Corkill, D. D. 1991. Blackboard systems. *AI Expert* 6(9):40–47. http://dancorkill.home.comcast.net/pubs/ai-expert.pdf.

Corkill, D. D. 2003. Collaborating software: Blackboard and multi-agent systems & the future. In *Proceedings of the International Lisp Conference*. http://dancorkill.home.comcast.net/pubs/ilc03.pdf.

Elmasri, R., and Navathe, S. B. 1999. *Fundamentals of Database Systems*. Boston: Addison Wesley, 4th edition.

Engelmore, R. S., and Morgan, A., eds. 1988. *Blackboard Systems*. Addison-Wesley.

Erman, L. D.; Hayes-Roth, F.; Lesser, V. R.; and Reddy, D. R. 1980. The hearsay-ii speech understanding system: Integrating knowledge to resolve uncertainty. *ACM Computing Survey* 12:213–253.

Galstyan, A., and Cohen, P. R. 2005. Identifying covert sub-networks through iterative node classification. In *Proceedings of the First International Conference on Intelligence Analysis*.

Gil, Y., and Ratnakar, V. 2002. Trellis: An interactive tool for capturing information analysis and decision making. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management*. www.isi.edu/ gil/papers/trellis-ekaw02.pdf.

Kolodner, J. L. 1993. *Case-based Reasoning*. California: Morgan Caufmann.

Levitt, R. E.; Cohen, G. P.; Kunz, J. C.; Nass, C. I.; Christiansen, T.; and Jin, Y. 1994. The 'virtual design

team': Simulating how organization structure and information processing tools affect team performance. In Carley, K. M., and Prietula, M. J., eds., *Computational Organization Theory*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Lowrance, J.; Harrison, I.; and Rodriguez, A. 2001. Capturing analytic thought. In *Proceedings of the First International Conference on Knowledge Capture*, 84–91. http://www.ai.sri.com/pubs/full.php?id=871.

Macskassy, S. A., and Provost, F. 2002. Simple models and classification in networked data. CeDER Working Paper 03-04, Stern School of Business, New York University. http://www.research.rutgers.edu/ sofmac/paper/ceder-03-04/abstract.html.

Maybury, M. 2001. Collaborative virtual environments for analysis and decision support. *Communications of the ACM* 44(12):51–54. http://www.mitre.org/work/tech-papers/tech-papers-01/maybury-collaborative/.

McCarthy, J. 1963. Situations, actions and causal laws. Technical Report Stanford Artificial Intelligence Project: Memo 2, Stanford University. http://wwwformal.stanford.edu/jmc/mcchay69/mcchay69.html.

Morrison, C. T.; Cohen, P. R.; King, G. W.; Moody, J. J.; and Hannon, A. 2005. Simulating terrorist threat with the hats simulator. In *Proceedings of the First International Conference on Intelligence Analysis*.

Neville, J., and Jensen, D. 2002. Supporting relational knowledge discovery: Lessons in architecture and algorithm design. In *Papers of the ICML 2002 Workshop on Data Mining Lessons Learned*. http://kdl.cs.umass.edu/papers/neville-jensen-dmll2002.pdf.

Newman, M. E. J. 2003. Fast algorithm for detecting community structure in networks. *Phys. Rev. E* 69(066133). http://aps.arxiv.org/abs/cond-mat/0309508/.

Nii, H. P. 1989. Blackboard systems. In Barr, A.; Cohen, P. R.; and Feigenbaum, E. A., eds., *The Handbook of Artificial Intelligence, Volume IV*. Addison-Wesley Publishing Company, Inc. chapter 17 (XVII), 1–82.

Nissen, M., and Levitt, R. E. 2004. Agent-based modeling of knowledge dynamics. *Knowledge Management Research & Practice* 2(3):169–183.

Shum, S. J. B.; Li, V. U. G.; Domingue, J.; and Motta, E. 2003. Visualizing internetworked argumentation. In Kirschner, P. A.; Sum, S. J. B.; and Carr, C. S., eds., *Visualizing Argumentation: Software Tools for Collaborative and Educational Sensemaking*. Springer-Verlag. chapter 9, 185–204. http://www.ecs.soton.ac.uk/∼ggl/publications/papers/VisNetArg2002.pdf.

Spellman, P. J.; Mosier, J. N.; Deus, L. M.; and Carlson, J. A. 1997. Collaborative virtual workspace. In *Proceedings of International ACM SIGGROUP Conference of Supporting Group Work, 16–19 November, Phoenix, Arizona*, 197–203. New York: ACM Press. See sourceforge.net/projects/cvw.

Sutton, C.; Burns, B.; Morrison, C. T.; and Cohen, P. R. 2003. Guided incremental construction of belief networks. In *Proceedings of the Fifth International Syposium on Intelligent Data Analaysis*. http://eksl.cs.umass.edu/papers/aiid-architecture-ida-03.pdf.

Sutton, C.; Morrison, C. T.; Cohen, P. R.; Moody, J.; and Adibi, J. 2004. A bayesian blackboard for information fusion. In *Proceedings of the 7th International Conference on Information Fusion*. http://eksl.cs.umass.edu/papers/fusion04.pdf.

van Gelder, T. J. 2002. Argument mapping with reason!able. In *The American Philosophical Association Newsletter on Philosophy and Computers*. http://www.arts.unimelb.edu.au/∼tgelder/papers/APA.pdf.

Watson, I., and Marir, F. 1994. Case-based reasoning: A review. *The Knowledge Engineering Review* 9(4):355–381. http://www.ai-cbr.org/classroom/cbr-review.html.

Weick, K. E. 1993. The collapse of sensemaking in organizations: The mann gulch disaster. *Administrative Science Quarterly* 38:628–652. http://www.myfirecommunity.net/documents/TheCollapseofSensemakinginOrganizations.pdf.

Wolverton, M.; Berry, P.; Harrison, I.; Lowrance, J.; Morley, D.; Rodriguez, A.; Ruspini, E.; and Thomere, J. 2003. Law: A workbench for approximate pattern matching in relational data. In *Proceedings of the Fifteenth Innovative Applications of Artificial Intelligence Conference (IAAI-03)*. http://www.ai.sri.com/pubs/full.php?id=931.