# The *Colab* Mixed-Initiative Analysis Enviroment

Clayton T. Morrison and Paul R. Cohen
Information Sciences Institute
University of Southern California
4676 Admiralty Way, Suite 1001
Marina del Rey, CA, 90292 U.S.A.
{clayton,cohen}@isi.edu

**Abstract -** ***COLAB is an analysis environment in which multiple human analysts in different physical locations can collaborate to build hypotheses of unfolding scenarios. COLAB consists of two components: an instrumented analysis working environment built on a blackboard system, and a web-based user interface that integrates the Trellis hypothesis authoring and management tool with a query language. On the blackboard, analysts collect data and perform analyses using relational database queries and a set of analysis tools that include group identification and suspicion scoring algorithms. Evidence extracted from data analysis on the blackboard may then be incorporated in hierarchical argument structures in the Trellis tool, combining evidence sources with user-supplied free text. Trellis arguments may then be shared between analysts and collaboratively authored. COLAB has been integrated with the Hats Simulator challenge domain and serves as a prototype mixed-initiative information fusion system.***

**Keywords:** Fusion architecture; application of fusion; sensor management; tracking and surveillance; command, control and operations research

## 1 Introduction

The COLAB Project brings together the Hats Simulator, a collaborative intelligence analysis environment, and a user interface to produce a prototype end-to-end system for intelligence analysis. Hats has been operational for three years and has been used in several studies, including providing data for part of the AFRL EAGLE project and assessing algorithms for relational data mining [1, 2]. The prototype intelligence analysis environment [3] and interface are implemented. The complete system has three intended applications:

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

Consider the following scenario. Several human analysts[1] work together in the COLAB environment to develop an interpretation of events in the Hats Simulator 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 the quality of their analyses.

In the following sections, we present the components of the COLAB system. We begin by very briefly describing the Hats Simulator. We then describe the blackboard system that serves as the integration model for 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 and agent definition language for accessing and manipulating information on the blackboard.

## 2 The Hats Simulator

The Hats Simulator poses a set of information fusion challenges that include integrating multi-source temporal data to infer group membership and structure, suspicion scoring, and plan inference. The Hats Simulator (described in [1, 2]) models a "society in a box" consisting of hundreds of thousands of simple agents, referred to as hats. A few hats are *known terrorists*; others are *covert* and must be identified and distinguished from the preponderance of *benign* hats. Before

[1] In the rest of this paper, we will use the terms "analyst" and "user" interchangeably to refer to a human analyst using the COLAB system.

a Hats game begins, the hats in the population are assigned to organizations, most of which are benign. All hats belong to multiple organizations and neither the organizations nor their members are known at the beginning of a game. All hats are governed by plans generated by a planner. Terrorist plans end in the destruction of *landmarks*. The planner constructs an elaborate "shell game" to transfer resources and skills through a series of meetings, culminating in a final meeting at a target. In this way the planner hides its intentions.

The object of a game against the Hats simulator is to minimize the sum of three costs. Through the Hats *Information Broker* interface, the player pays for information about hats, and more accurate information costs more. Players are also assessed costs for making false arrests and not stopping terrorists before they attack landmarks. At the end of a game, the player's score is the sum of these costs. In general, one cannot reduce the costs of successful attacks without increasing the costs of information and false arrests.

COLAB has been interfaced with the Hats Information Broker. The interface defines a set of basic information request types that may be initiated from within COLAB and sent to the Hats Information Broker. The eight request types are: the hats (if any) at a location, participants in a meeting, capabilities carried by a hat, capability trades, meeting times, hat death time, meeting locations and hat locations.

# 3 The COLAB Analysis Environment

The Hats game is very challenging. There is a vast amount of information to keep track of. There are hundreds of thousands of entities. The properties of these entities change from one time step to the next. There is considerable hidden structure, including organization membership, task force membership, coordinated task goals, and benign or malevolent intention. Identifying any of this structure will require keeping track of individual and group behavioral histories. Interacting with the Information Broker alone makes the task nearly impossible. Any analysis environment worth its salt must help the analyst manage and explore this information. This is the goal of COLAB analysis environment.

The COLAB analyst environment is built on a blackboard architecture designed to represent relational data and integrate a variety of intelligence analysis algorithms as problem solving components. We first introduce the blackboard system architecture and why we believe it is well-suited for this kind of task, and then describe the COLAB blackboard components.

## 3.1 Blackboard Systems

Blackboard systems are knowledge-based problem solving environments that work through the collaboration of independent reasoning modules [4, 5]. More recently blackboards have been recognized as platforms for data fusion [6]. They were developed in the 1970s and originally applied to signal-processing tasks. The first, HEARSAY-II [7], 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 [4].

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 problems. 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* (`KS`s) are modular, self-contained programs which post results of local computations to the blackboard. Different `KS`s 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* [8]. Generally it works like this: `KS`s watch for particular kinds of results on the blackboard (a *trigger condition*); 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 `KSAR`s 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.

The blackboard architecture is ideal for prototyping an intelligence analysis and information fusion system. First, blackboard processing is opportunistic. Intelligence information arrives at different times, out of order and in varying quality and quantity. Blackboards allow "islands" of hypotheses in varying stages of processing to exist simultaneously. These islands are augmented as new information becomes available and at any time they have the potential to contribute to the overall representation of the situation, even if they are incomplete. Second, blackboards allow both bottom-up (data-driven) and top-down (hypothesis and knowledge-driven) processing. Data from the environment can trigger `KS`s to steer attention to new events, while at the same time knowledge from higher levels may be brought to bare, potentially biasing search and lower-level evidence evaluation. Finally, blackboards are an integration architecture: knowledge sources can encapsulate many different functions, knowledge, and reasoning, only participating when their preconditions are met. We can wrap a wide variety of different analysis tools in a `KS` and define the appropriate conditions under which they should do their

work. Taken together, the properly configured blackboard architecture supports distributed, opportunistic intelligence analysis that can incorporate multiple different analysis algorithms, including human analysts.

## 3.2 The COLAB Blackboard

The COLAB blackboard follows the same architectural principles of a standard blackboard except that human users interact with the blackboard, playing the same role as other domain and control knowledge sources. The COLAB blackboard is the analyst's "workspace," representing and managing information that the analyst gathers from the the Hats Information Broker. In this workspace, the analyst views, manipulates and processes Information Broker reports to extract evidence supporting hypotheses about ongoing on potential future events. Analysts are also provided an interface to define and manage their own KS assistants, *agents* that can handle routine querying, watching for user-defined conditions of interest, and sending alerts or reminders when conditions are satisfied.

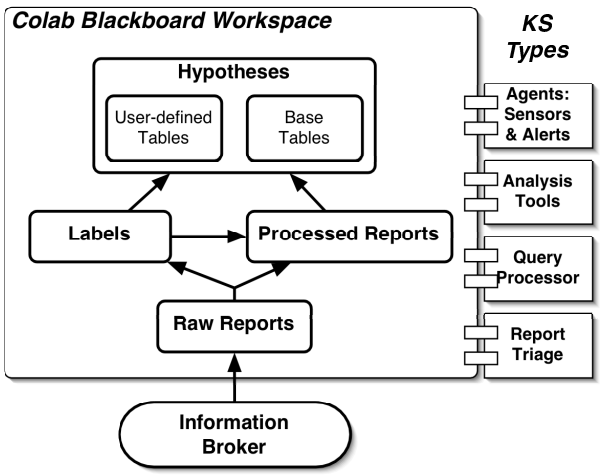


Figure 1: The COLAB blackboard. Each box on the blackboard represents a blackboard space; the *Hypothesis* space contains two subspaces. The oval at the bottom of the figure represents the Information Broker (the interface to the Hats Simulator), and the tabs to the right represent types of KSs. Arrows indicate flow of information between spaces that results from knowledge source processing.

### 3.2.1 Blackboard Spaces

Figure 1 depicts the COLAB blackboard and its component spaces. The Raw Reports, Processed Reports and Labels spaces are responsible for representing reports from the Hats Information Broker as atomic assertions about entities and relations in the Hats domain. For example, the analyst may request from the Information Broker information about the members of a meeting hypothesized to have taken place. In response, the Information Broker returns a report that is then posted to the *Raw Reports* space.

This single report represents several entities and atomic relations: the set of meeting participants, the meeting event, the time and location of the meeting, and also the individual assertions that the participating hats were at that location at that time. COLAB *report triage* knowledge sources decompose the report into these atomic assertions so that the analyst and other KSs will be able to refer to each component individually and manipulate and combine the components in novel ways. For example, the analyst may want to later ask how many meetings a given hat was in over a period of time; or she may only want to know where that hat was at that time, irrespective of what meeting the hat was in. In Section 4.2 we describe the query language that makes this possible.

Next, each atomic assertion is 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 the reliability of the information derived from the report.

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 part in the meeting from the example above, then a label for $Hat_{27}$ is linked to two atomic reports, one asserting that it participated in the meeting and the other asserting that it was at that location at that time. (In the same way, there is a label representing 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.

The *Hypotheses* space of the COLAB blackboard is reserved for representing analyst work. All contents of this space are represented as relational database tables. A wide variety of data and knowledge representation schemes is available, many more powerful than the basic relational database model. However, we have chosen the relation model as the standard data representation because the format for storing data is simple, it provides a relatively small set of operators for data manipulation, and the model has wide use and a long successful history [9].

*Labels* on the Labels space serve to define *base tables* in the Hypothesis space. These base tables are an index into the space of processed reports and are used in query processing and Hypothesis space browsing. The results of all user-initiated queries are stored in the User-defined Tables space.

The blackboard schematic of Figure 1 represents the blackboard workspace for a single human user. In multi-user configurations, each user has their own workspace blackboard structure in the same configuration as Figure 1. *Shared* blackboard spaces are then provided as hypothesis spaces that multiple users can view, post and edit user-created relational database tables. This makes it possible for users to share analyses in the form of evidence represented in relational tables. In Section 4.1, we also describe how argument

structures incorporating this evidence may be shared through the Trellis interface.

#### 3.2.2 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 consists of KS interfaces or wrappers for 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 [6]; as long as these algorithms can read representations on the blackboard and write output to the blackboard that is interpretable by other KSs, they can participate in blackboard processing. Some examples of analysis services include algorithms for assigning suspicion scores to hats [10, 11], identifying community structure [12, 13], 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 Hats Information Broker.

We are currently prepared to offer an implementation of Mark Newman's modularity-based community finding algorithm [13] as an example 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.

Another class of KSs consists of the user-defined "agents" that perform simple tasks. Like other KSs, agents have a trigger condition and action that is taken if the condition is met, but triggers and actions are restricted to what can be expressed in components of the COLAB query language, described in Section 4.2. These agents may issue there own queries, check for conditions, and either trigger other KSs or send messages to the users. They may 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.

#### 3.2.3 Control

Current blackboard control in COLAB is basic, consisting of a priory queue-based agenda shell. Report and query processing has 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 [14]. 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, providing suggestions to the analyst for how much to spend on a query, or automatically adjusting cost of automated queries. In general, we treat control as an open development issue in COLAB that will be shaped by performance with multiple users and increasing data and processing loads.

## 4 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 web-based 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. Figure 2 shows a snapshot of the COLAB web interface running in the free Firefox[3] web browser.

The goal of COLAB interface design is intuitive 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 start with existing technologies. For hypothesis representation, COLAB uses the Trellis argument authoring tool. For information access we have implemented a subset of standard SQL with small extensions. And our initial interface for browsing blackboard contents will be hypertext-based.

### 4.1 Trellis

Trellis [15] is an interactive web-based application for argumentation and decision-making. Trellis has its roots in the Knowledge Capture AI community, whose aim is to make it possible for unassisted users to represent their knowledge in a form that can be use by knowledge-base AI systems. Trellis takes a step in this direction by allowing the user to express herself using a combination of semi-formal argument terms and relations to structured combinations of free text and documents found on the web (including text, images, video and other media). Users individually or collaboratively author structured arguments and analyses. The tool

[2]Macintosh Common Lisp 5.1 running on Macintosh OS X 10.3.9

[3]http://www.mozilla.com/firefox/

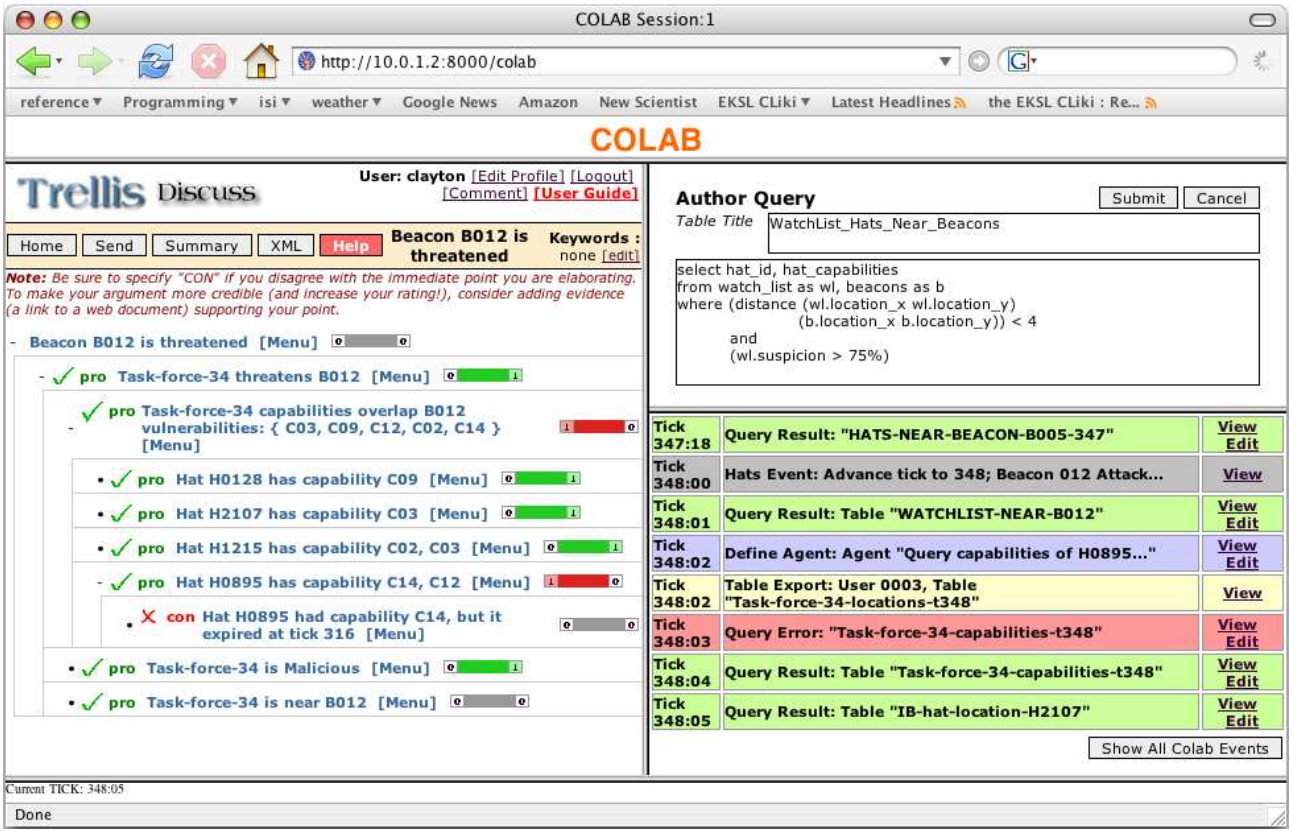


Figure 2: The COLAB web interface. The left half of the browser window contains the interface to the Trellis argument authoring tool. An argument about whether beacon B12 is threatened is currently being edited. The upper right of the browser window contains the field for entering commands and queries. The Author Query command template is currently selected. The bottom right provides a history of COLAB events, including events created by user input, events initiated by other users (such as making a table public) and Hats Simulator world-state events, such as advancing a tick.

is domain independent, permitting general structured argumentation about any topic.

Trellis analyses can be built using several different formats. A Trellis analysis involves constructing a tree structure, where each "node" in the tree is a statement, consisting of free text and evidence sources (including web documents). Composite statements are formed using semi-formal links such as 'causes', 'depends on', 'has attributes' and 'is elaborated in'. Analyses themselves may also be linked with one another. An analysis is typically authored by a single user, but it may be exported or made public so that other analysts can use or comment on it.

A Trellis *discussion* is also structured as a tree of Trellis statements, but parent/child links are restricted to pro/con relationships, expressing that the child statement either supports or contradicts the parent. Multiple users may vote on a discussion statement to record whether they agree or disagree with it. A tally is kept of the votes to indicate whether a statement is believed by the group to be true, false or contentious.

Evidence associated in any of the above Trellis structures is gathered from source documents on the web. An interface allows the user to select an entire web document or portions of a web page. Sources are collected in a database for later possible inclusion in analyses. The Trellis web interface makes it an attractive, lightweight and powerful tool for conducting individual and collaborative analyses. The ability to associate free text with semi-formal terms and connectives is intuitive and also allows user input to have a more machine-readable form. Also, being designed for general web document inclusion, Trellis is already capable of open source intelligence analysis authoring.

We have chosen to adopt Trellis as the COLAB hypothesis authoring tool because it expresses the basic hierarchical support/dissent relationships between statements that we believe will be useful to analysts. Trellis also provides a well-designed, intuitive user interface. The left half of the COLAB browser window in Figure 2 shows an example of the Trellis Discussion interface. The interface currently displays an argument supporting the hypothesis that beacon `B012` is threatened. The top-level statement (the "target hypothesis") asserts that `Task-force-34` threatens beacon `B012`. This claim is supported by three additional statements, that `Task-force-34` has capabilities that overlap the vulnerabilities of the beacon, that `Task-force-34` is malicious, and that the members of `Task-force-34` are near the beacon. Capability overlap is further elaborated by evidence that the capabil-

ities 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 `H0895` is no longer carrying the remaining required capability `C14`, having apparently expired at tick 316. The Trellis window also represents, by the colored bars to the right of each statement, a tally of votes provided by other analysts indicating whether they agree or disagree with the statement. In this case, only one other analyst has voted; green indicates they agree, red indicates they do not agree with the statement.

## 4.2 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. In order to gather, manipulate and manage this data, we have implemented a subset of SQL, an intuitive, well-studied language with a long, successful history in practical database applications [9]. In the relational model, an individual fact is represented as a set of attribute values. Facts are collected in tables in which each row represents a fact, each column corresponds to an attribute, and the cell at the row/column intersection holds the the attribute value for the fact. Queries specify what information to extract from tables, and the results of queries are expressed in new tables. In the COLAB blackboard, hypothesis spaces are treated as tables and all queries result in tables stored on the (relational) *User-define Tables* space of the blackboard.

The upper-right frame of the browser window in Figure 2 shows the query entry field. The query example in the window requests a table containing the `hat_id`s and `hat_capabilities` for hats from the `watch_list` table – but only for those hats that are within distance 4 of a beacon and have a suspicion greater than 75%. The table has also been assigned the name `WatchList_Hats_Near_Beacons` by the user.

```
select giver, taker, capability,
       meeting, tick
from suspicious_trade_reports
where (between tick (current_tick - 10)
               and current_tick);
```

Figure 3: Example query to the Workspace retrieving all trade reports in the suspicious_trade_reports table the past 10 ticks

Because all reports are indexed by time, COLAB also provides facilities for specifying individual time intervals. Figure 3 shows an example query that uses the special constant `current_time` to request all reported trade events in the past 10 ticks. The requested table includes attributes representing the hat that gave the capability, the hat that received the capability, the capability itself, the label representing the meeting in which the trade took place, and the time when the trade took place.

```
select hat_location
from ib
where hat_id = (select hat_id
                from watch_list)
      and payment = 2000;
```

Figure 4: Example query to the Information Broker asking for location of all hats on the `watch_list`.

Finally, Figure 4 shows how to use the same query language to send a request to the Information Broker. The user specifies in the `from` clause the source `information_broker` (or `ib` for short). The `select` clause, rather than specifying the names of attributes to retrieve, instead specifies one of the eight predefined Information Broker report types. Finally, the `where` condition sets the argument values for the `hat_location` report type. Queries to the Information Broker may be augmented by queries to the Hypothesis space. The query in Figure 4 demonstrates this with a sub-query used to first select the `hat_id`s of all hats on the `watch_list` table and then pass those as arguments to the outer `hat_location` Information Broker query. In this way the user can ask for all `hat_location` reports at once, rather than specifying each location report separately for each hat.

## 4.3 Agents: User-defined KSs

Knowledge sources provide another facility for analyst information management. As mentioned in Section 3.2.2, users may use the query language to specify `KS` triggers and actions. When the user defines an agent, they are presented with a template for specifying agent settings. The three main components are the agent activation condition, the trigger condition and the action.

An activation condition has two components. The user enters a natural number in the `schedule` field to specify the frequency with which the agent will be triggered. For example, entering 3 means the agent will run every three ticks: on one tick, then sleeps for two before being triggered again. Entering 0 means the agent won't be triggered on a schedule. The second field allows the user to specify `KS activation messages`, a list of zero or more user-defined names[4] that may be posted to the blackboard by other user-defined agent `KS`s. When another `KS` posts any one of the matching trigger message to the blackboard, this agent will then be triggered. One or both of the activation conditions may be specified. A combination entails that the agent will be activated on a regular schedule, and perhaps sooner if a matching activation message is posted.

Finally, the `predicate` field determines whether, when activated, the action will be taken. Since the

[4] A name can be any string of letters following the same rules for naming a table: any combination of upper or lower case letters, underscores or numbers that are together do not form a reserved word; analysts are provided with a list of COLAB reserved words.

`where` clause of a standard query is used to express a statement that is either true or false depending on values, trigger predicates are specified using the same clause syntax. The predicate is optional; if left blank, an activated agent will always execute its action.

The action component of the agent consists of any combination of (1) a standard blackboard or Information Broker query, (2) a COLAB event message, or a `KS activation message`. When a query is specified, it is executed when the agent action is taken. A COLAB event message is a string that is posted to the COLAB Event History, alerting the user to the agent's triggering condition having been satisfied. And the `KS activation message` will be posted to the blackboard, possibly triggering other user-defined agents.

The following is a set of example tasks that can be expressed as agents, to help with routine monitoring of events in the Hats Simulator:

- **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 agent may be defined to trigger when the meeting alert is issued; this sensor may then send a 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 Watchlist table.
- **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.

Using the agent specification template, the analyst can build up a toolkit of useful, special purpose knowledge source, blurring the line between human control and blackboard automation.

### 4.4 Blackboard Browsing

The Trellis hypothesis authoring tool and the query language allow the analyst to visualize relations between hypotheses and evidence stored as relational tables on the blackboard. Relational tables are viewed a HTML generated tables, and these are incorporated as evidence linked to a Trellis statement by using the Trellis web page content extraction tool.

The user is also provided with indexes of blackboard contents, such as the list of tables currently defined on the User-defined Hypothesis space, or on the Base-tables space. The user may also view the current status of user-defined agents and have the option to turn them on or off.

Finally, to help keep track of Hats world and COLAB events, the lower-left window of the COLAB interface keeps a history of all events. Each event is given one line in the history. The left-hand cell gives the current tick, which consists of the Hats world-state tick and the current user-event number for that tick. For example, the grey bar in the event history window of Figure 2 denotes the Hats world event that the tick has advanced to 348 and a new set of Hats events are reported, one of which is the attack of Beacon 012. The next event, during tick 348, the first event initiated by the user, is a query that has resulted in the table named "Query capabilities of H0895" (quoted names have implicit underscores for spaces). This was followed by the event that another user has made public a new table on a shared workspace. This, in turn, is followed by an attempted query by the user that has failed because of an error. It is expected that not all command entry will be correct and COLAB attempts to help the user identify the problem. On the right-hand column are hyperlinks to view or edit Hats events details, tables that result from queries or posts by other users, agent definitions, and any failed query attempts. When the edit link is available and selected, the original entered command text is provided and the user may make changes and resubmit the query or new, modified agent.

### 4.5 Implementation

COLAB is was developed in Macintosh Common Lisp (MCL) 5.1 (http://www.digitool.com/) and openMCL 1.0 (http://www.openmcl.org/) running on Macintosh OS X. For the blackboard we used the GBBopen blackboard framework (http://gbbopen.org/). The web interface server is also written in Common Lisp, built on top of the lightweight Araneida server (http://www.cliki.net/araneida). Except for MCL and Mac OS X, all of these software packages are open source.

## 5 Concluding Remarks

In the introduction we described three target uses for COLAB: an environment for studying analyst collaboration and analysis tools, an environment for training analysts, and a configurable laboratory to study varying command and control structures for network-centric, distributed intelligence analysis. All three are very ambitious goals and COLAB is still an early prototype. However, the facilities described in this paper are implemented and we are beginning user testing. COLAB provides and interesting, novel information fusion architecture for intelligence analysis that is unique in its pairing of a hypothesis authoring tool, provided in Trellis, a configurable multi-user information man-

agement system, provided by the COLAB blackboard, and a challenging, discrete-time simulated problem domain that can be played online.

## 6 Acknowledgements

Work on this project was 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. Yolanda Gil, Dr. Tim Chklovski and Varun Ratnakar for discussions and help with integrating COLAB with Trellis, and Dr. Dan Corkill and Gary W. King for help with GBBopen and Ijara. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

## References

[1] P. R. Cohen and C. T. Morrison. The hats simulator. In *Proceedings of the 2004 Winter Simulation Conference*, 2004.

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

[3] C. T. Morrison and P. R. Cohen. Colab: A laboratory environment for studying analyst sensemaking and collaboration. In *Proceedings of the Tenth International Command and Control Research and Technology Symposium (10th ICCRTS)*, 2005.

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

[5] D. D. Corkill. Blackboard systems. *AI Expert*, 6(9):40–47, September 1991.

[6] D. D. Corkill. Collaborating software: Blackboard and multi-agent systems & the future. In *Proceedings of the International Lisp Conference*, New York, October 2003.

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

[8] N. Carver and V. Lesser. The evolution of blackboard control architectures. *Expert Systems with Applications–Special Issue on the Blackboard Paradigm and Its Applications*, 7(1):1–30, January 1994.

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

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

[11] S. A. Macskassy and F. Provost. Simple models and classification in networked data. CeDER Working Paper 03-04, Stern School of Business, New York University, 2002.

[12] J. Adibi, P. R. Cohen, and C. T. Morrison. 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)*, 2004.

[13] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Phys. Rev. E*, 69(066133), 2003.

[14] C. T. Morrison and P. R. Cohen. Noisy information value in utility-based decision making. In *Proceedings of the First International Workshop on Utility-Based Data Mining (UBDM SIGKDD 2005)*, pages 34–38, 1515 Broadway, New York, New York 10036, 2005. The Association for Computing Machinery, Inc.

[15] T. Chklovski, V. Ratnakar, and Y. Gil. User interfaces with semi-formal representations: a study of designing argumentation structures. In *Under Review for the Intelligent User Interfaces Conference 2005*, 2005.