The Hats Simulator and Colab: An Integrated Information Fusion Challenge Problem and Collaborative Analysis Environment

Clayton T. Morrison and Paul R. Cohen

Center for Research on Unexpected Events (CRUE)
Information Sciences Institute
University of Southern California
4676 Admiralty Way, Suite 1001
Marina del Rey, California, USA
{clayton,cohen}@isi.edu
http://crue.isi.edu/

Abstract. We present an overview of our work in information fusion for intelligence analysis. This work includes the Hats Simulator and the COLAB system. The Hats Simulator is a parameterized model of a virtual world in which hundreds of thousands of agents engage in individual and collective activities. Playing against the simulator, the goal of the analyst is to identify and stop harmful agents before they carry out terrorist attacks on simulated landmarks. The COLAB system enables multiple human analysts in different physical locations to conduct collaborative intelligence analysis. 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. COLAB is integrated with the Hats Simulator to provide a complete end-to-end analysis environment with challenging problem domain.

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 Hats Simulator and the components of the COLAB system. We begin in Section 2 by describing the Hats Simulator, including the Hats domain, the Information Broker interface, and scoring. In Section 3 describe the blackboard system that serves as the integration model for the COLAB analysis working environment, the web-based interface to COLAB that incorporates the Trellis argument authoring 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 [1, 2] is home to hundreds of thousands of agents (hats) which travel to meetings. Some hats are covert terrorists and a very few hats are known terrorists. All hats are governed by plans generated by a planner. Terrorist plans end in the destruction of landmarks. The object of a game against the Hats simulator is to find terrorist task forces before they carry out their plans. One pays for information about hats, and also for false arrests and destroyed landmarks. At the end of a game, one is given a score, which is the sum of these costs. The goal is to play Hats rationally, that is, to catch terrorist groups with the least combined cost of information, false arrests, and destroyed landmarks. Thus Hats serves as a testbed not only for analysts' tools but also for new theories of rational information fusion that take into account information source assessment and cost in the context of analysis goals and decisions. Hats encourages players to ask only for the information they need, and to not accuse hats or issue alerts without justification.

[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.

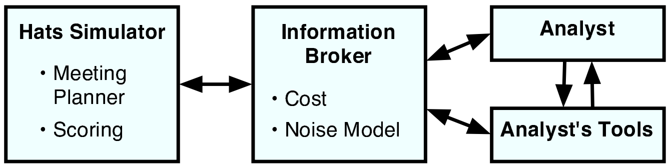


Fig. 1. Information Broker Interface to the Hats Simulator

The Hats Simulator consists of the core simulator and an Information Broker. 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 (see Figure 1). Some information has a cost, and the quality of information returned is a function of the "algorithmic dollars" spent. Analysts may also take actions: they may raise beacon alerts in an attempt to anticipate a beacon attack, and they may arrest agents believed to be planning an attack. Together, information requests and actions form the basis of scoring analyst performance in identifying terrorist threats. Scoring is assessed automatically and serves as the basis for analytic comparison between different analysts and tools. The simulator is implemented and manages the activities of hundreds of thousands of agents.

2.1 The Hats Domain

The Hats Simulator models a "society in a box" consisting of many very simple agents, hereafter referred to as 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.

Population Generation Hats populations may be built by hand or generated by the Hats Simulator. Because the constitution of a population affects the difficulty of identifying covert terrorists, population generation is parameterized. There are four sets of population parameters. The first set specifies the total number of known terrorists, covert terrorists and benign hats in the population. Another set defines the number of benign and terrorist organizations. Not all organizations have the same number of members, so a third set of parameters assigns the relative numbers of hats that are members of each organization, represented as a ratio among organizations. Finally, hats may be members of two or more organizations. An overlap parameter determines the percentage of hats in each organization that are members of two or more other organizations. Since hat behaviors are governed by their organization membership, as we will see in the next section, organization overlap affects how difficult it is to identify covert terrorist hats. To generate populations with hundreds of thousands of hats and thousands of organizations, we use a randomization algorithm that estimates organization overlap percentage and membership ratios while matching the total number of organizations and hats in the population. An efficient, approximate population generation algorithm is described in [4].

Meeting Generation 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 passed 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 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.

Each organization has a generative planner that plans tasks for its members. At each tick, each organization has a chance of beginning a new task. When a new task is started, the Hats meeting planner creates a task force, the size of which is controlled by a parameter. The planner next selects a target location in the Hats world. If a beacon is selected as the target, the goal of the task is to bring to that location the set of required capabilities that match the vulnerabilities of the beacon. If the location is not a beacon, a random set of required capabilities is selected as the set to bring to the location.

Task force members may or may not already possess the required capabilities; usually they dont. The planner creates a set of meetings designed to ensure that the task force acquires all of the required capabilities before going to the target location. This is accomplished by constructing a meeting tree that specifies meetings and their temporal order. Figure 2 shows an example meeting tree, where boxes represent planned meetings among hats and arrows represent the planned temporal partial order of meetings. The tree is "inverted" in the sense

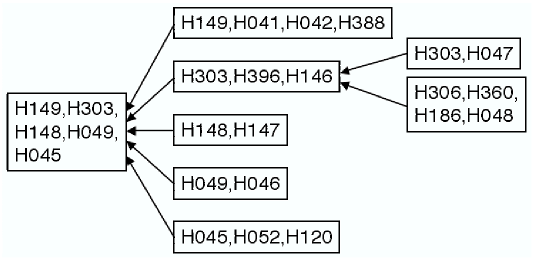


Fig. 2. Example of a generated meeting tree. Each box represents a meeting and contains a list of participating hats. Arrows indicate planned temporal order of meetings.

that the arrows point from leaves inward toward the root of the tree. Parent meetings, where arrows originate, are executed first. When all of the parent meetings of a child meeting have completed, then the child meeting happens.

2.2 The Information Broker

As an analyst playing the game, your job is to protect the Hats society from terrorist attacks. You need to identify terrorist task forces before they attack beacons, but you also need to avoid falsely accusing innocent hats. The only way to do this successfully is to gather information about hats, identify meetings, track capability trades and form hypotheses about the intentions of groups of hats. 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}."

Some information is free, but information about states of the simulator that change over time is costly. The quality of the information obtained is determined by the amount paid.

Eight basic information requests can may be made: the hats at the location (if any), participants in a meeting, capabilities carried by a hat, capability trades, meeting times, hat death time, meeting locations and hat locations. The first five requests return lists of things, such as hats, capabilities, times, etc. The latter three are scalar values.

2.3 Actions

In addition to requesting information, the analyst playing the Hats game can also change a beacon's alert level and arrest hats. Both actions affect an analyst's performance score (discussed in Section 2.4).

Raising Alerts 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 *beacon 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 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.

Arresting Hats 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.

2.4 Scoring Analyst Performance

The Hats Simulator and Information Broker together provide an environment for testing analyst tools. The object of the game is to identify terrorist task forces before they attack beacons. 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.

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 [5, 6]. More recently blackboards have been recognized as platforms for data fusion [7]. They were developed in the 1970s and originally applied to signal-processing tasks. The first, HEARSAY-II [8], 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 [5].

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* [9].

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 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.

3.3 COLAB Knowledge Sources

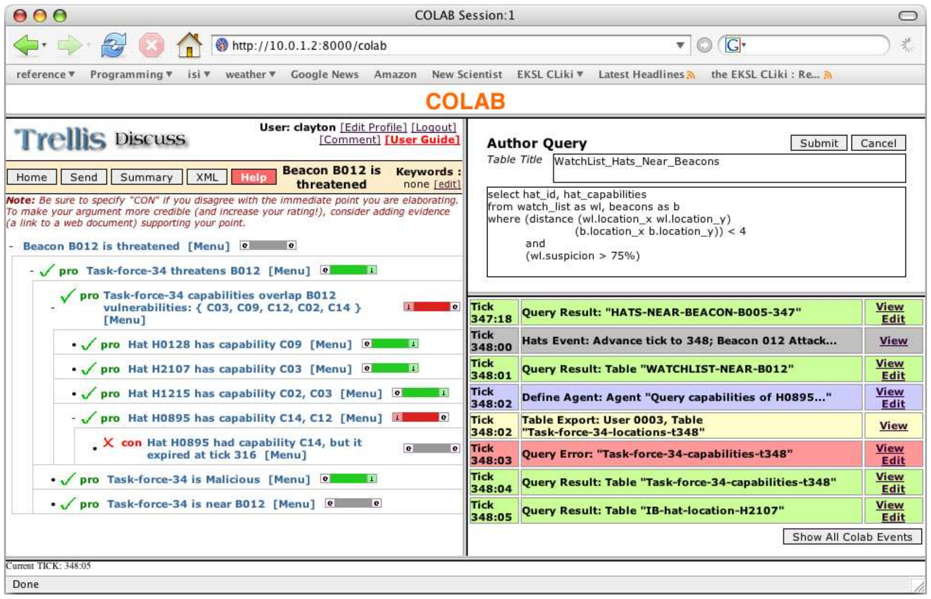
A class of `KS`s called *report triage* `KS`s handle processing information broker reports and updating the Labels and Processed Reports spaces. Another class consists of KS interfaces or wrappers for algorithms available to the analyst as analysis tools or "services." The beauty of the blackboard architecture is that it is specifically designed to facilitate collaborative software [7]; 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], identifying community structure [11, 12], 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.

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 3.4. 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.

3.4 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 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.

Trellis Trellis [13] 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 is domain independent, permitting general structured argumentation about any topic.


COLAB Session:1
http://10.0.1.2:8000/colab
reference ▾ Programming ▾ isi ▾ weather ▾ Google News Amazon New Scientist EKSL CLiki ▾ Latest Headlines the EKSL CLiki : Re...
COLAB
Trellis Discuss
User: clayton [Edit Profile] [Logout]
[Comment] [User Guide]
Home Send Summary XML Help
Keywords : none [edit]
Author Query
Submit Cancel
Show All Colab Events
Done

Beacon B012 is threatened

Note: Be sure to specify "CON" if you disagree with the immediate point you are elaborating. To make your argument more credible (and increase your rating!), consider adding evidence (a link to a web document) supporting your point.

- Beacon B012 is threatened [Menu]
  - ✓ pro Task-force-34 threatens B012 [Menu]
    - ✓ pro Task-force-34 capabilities overlap B012 vulnerabilities: { C03, C09, C12, C02, C14 } [Menu]
      - ✓ pro Hat H0128 has capability C09 [Menu]
      - ✓ pro Hat H2107 has capability C03 [Menu]
      - ✓ pro Hat H1215 has capability C02, C03 [Menu]
      - ✓ pro Hat H0895 has capability C14, C12 [Menu]
        - ✗ con Hat H0895 had capability C14, but it expired at tick 316 [Menu]
    - ✓ pro Task-force-34 is Malicious [Menu]
    - ✓ pro Task-force-34 is near B012 [Menu]

Table Title: WatchList_Hats_Near_Beacons

```
select hat_id, hat_capabilities
from watch_list as wl, beacons as b
where (distance (wl.location_x wl.location_y)
               (b.location_x b.location_y)) < 4
      and
      (wl.suspicion > 75%)
```

Tick 347:18	Query Result: "HATS-NEAR-BEACON-B005-347"	View Edit
Tick 348:00	Hats Event: Advance tick to 348; Beacon 012 Attack...	View
Tick 348:01	Query Result: Table "WATCHLIST-NEAR-B012"	View Edit
Tick 348:02	Define Agent: Agent "Query capabilities of H0895..."	View Edit
Tick 348:02	Table Export: User 0003, Table "Task-force-34-locations-t348"	View
Tick 348:03	Query Error: "Task-force-34-capabilities-t348"	View Edit
Tick 348:04	Query Result: Table "Task-force-34-capabilities-t348"	View Edit
Tick 348:05	Query Result: Table "IB-hat-location-H2107"	View Edit

Current TICK: 348:05

Fig. 3. 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.

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 3 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 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 `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.

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 [14]. 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. The upper-right frame of the browser window in Figure 3 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.

Agents: User-defined KSs Knowledge sources provide another facility for analyst information management. As mentioned in Section 3.3, users may use the query language to specify `KS` triggers and actions.

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.

3.5 COLAB Implementation

COLAB is was developed in Macintosh Common Lisp[2] (MCL) 5.1 and open-MCL[3] 1.0 running on Macintosh OS X. For the blackboard we used the GB-Bopen[4] blackboard framework. The web interface server is also written in Common Lisp, built on top of the lightweight Araneida[5] server. Except for MCL and Mac OS X, all of these software packages are open source.

4 Concluding Remarks

In the introduction we described three target uses for the Hats Simulator in conjunction with 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 management system, provided by the COLAB blackboard, and a challenging, discrete-time simulated problem domain that can be played online.

5 Acknowledgements

The Hats Simulator was conceived of by Paul Cohen and Niall Adams at Imperial College in the summer of 2002. Dr. Cohen implemented the first version of Hats. Work on the Hats Simulator was funded by the Air Force Research Laboratory, account number 53-4540-0588. Work on the COLAB 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.

[2] http://www.digitool.com/
[3] http://www.openmcl.org/
[4] http://gbbopen.org/
[5] http://www.cliki.net/araneida

References

1. Cohen, P.R., Morrison, C.T.: The hats simulator. In: Proceedings of the 2004 Winter Simulation Conference. (2004)
2. Morrison, C.T., Cohen, P.R., King, G.W., Moody, J.J., Hannon, A.: Simulating terrorist threat with the hats simulator. In: Proceedings of the First International Conference on Intelligence Analysis. (2005)
3. Morrison, C.T., Cohen, P.R.: 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. Hannon, A.C., King, G.W., Morrison, C.T., Galstyan, A., Cohen, P.R.: Population generation in large-scale simulation. In: Proceedings of AeroSense 2005. (2005)
5. Nii, H.P.: Blackboard systems. In Barr, A., Cohen, P.R., Feigenbaum, E.A., eds.: The Handbook of Artificial Intelligence, Volume IV. Addison-Wesley Publishing Company, Inc. (1989) 1–82
6. Corkill, D.D.: Blackboard systems. AI Expert **6**(9) (1991) 40–47
7. Corkill, D.D.: Collaborating software: Blackboard and multi-agent systems & the future. In: Proceedings of the International Lisp Conference, New York (2003)
8. Erman, L.D., Hayes-Roth, F., Lesser, V.R., Reddy, D.R.: The hearsay-ii speech understanding system: Integrating knowledge to resolve uncertainty. ACM Computing Survey **12** (1980) 213–253
9. Carver, N., Lesser, V.: The evolution of blackboard control architectures. Expert Systems with Applications–Special Issue on the Blackboard Paradigm and Its Applications **7**(1) (1994) 1–30
10. Galstyan, A., Cohen, P.R.: Identifying covert sub-networks through iterative node classification. In: Proceedings of the First International Conference on Intelligence Analysis. (2005)
11. Adibi, J., Cohen, P.R., Morrison, C.T.: 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)
12. Newman, M.E.J.: Fast algorithm for detecting community structure in networks. Phys. Rev. E **69**(066133) (2003)
13. Chklovski, T., Ratnakar, V., Gil, Y.: User interfaces with semi-formal representations: a study of designing argumentation structures. In: Under Review for the Intelligent User Interfaces Conference 2005. (2005)
14. Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems. 4th edn. Boston: Addison Wesley (1999)