LAW: A Workbench for Approximate Pattern Matching in Relational Data

Michael Wolverton, Pauline Berry, Ian Harrison, John Lowrance
David Morley, Andres Rodriguez, Enrique Ruspini, Jerome Thomere
Artificial Intelligence Center
SRI International
333 Ravenswood Ave
Menlo Park, California 94025
<*lastname*>@ai.sri.com

Abstract

Pattern matching for intelligence organizations is a challenging problem. The data sets are large and noisy, and there is a flexible and constantly changing notion of what constitutes a match. We are developing the Link Analysis Workbench (LAW) to assist an expert user in the intelligence community in creating and maintaining patterns, matching those patterns against a large collection of relational data, and manipulating partial results. This paper describes two key facets of the LAW system: (1) a pattern-matching module based on a *graph edit distance* metric, and (2) a system architecture that supports the integration and tasking of multiple pattern matching modules based on their capabilities and the specific problem at hand.

Introduction

An important role of intelligence organizations is to identify and track situations of interest—terrorist and other criminal activity, signs of impending political upheaval abroad, etc.—in a sea of noisy and incomplete information. A large amount of the information intelligence professionals make use of today is in relational form (i.e., stored in relational databases), and we can expect that amount to increase dramatically in the near future as information extraction and other technologies for converting unstructured data sources into structured ones mature and see greater use. There is a critical need for technology that helps intelligence experts find and elaborate evidence in relational data. Specifically, we want to develop tools that help an intelligence analyst define and match patterns in relational data, where the notion of "match" is very broad and gives the analyst a large amount of flexibility.

We are developing the Link Analysis Workbench (LAW) to help meet this need. LAW is a system designed to allow a domain expert user to build and refine patterns quickly, to search for matches from a large data set, and to understand and compare the matches it finds easily. Because the intelligence domain involves missing data, erroneous data, and even imprecise user understanding of what the right pattern should be, LAW is specifically focused on *approximate* pattern matches—situations in the data that are similar to, but not exactly the same as, the situation represented in the pattern. LAW is also designed as a user-centric system, where the pattern representation and matching criteria are understandable by an intelligence expert user, and where the user can play an important, hands-on role in the system's cycle of authoring, matching, and revising patterns.

This paper is organized around two major aspects of the LAW system:

- A graph-based pattern representation and matching capability. This includes a flexible, hierarchical pattern representation language based on graphs, a pattern comparison metric based on a variant of *graph edit distance*, and an anytime search mechanism for finding approximate matches to the pattern in large data sets.
- An architecture and user interface that support integration of multiple pattern matching tools, including LAW's own. This architecture allows the user to specify patterns in a generic, user-friendly framework, allows patterns to be dispatched to the matching tool(s) most appropriate for it, and merges the resulting matches and presents them to the user in a common framework.

We begin by describing LAW's pattern representation and pattern matching approach. Then we give an example of LAW's pattern matching behavior and describe LAW's user interface. After that, we describe the architecture and common pattern language LAW uses to integrate multiple pattern matching tools into a single system. Finally, we describe open issues and planned future work on the system.

Patterns and Matching

The goal of the LAW pattern matching component is to help intelligence analysts[1] find instances in the world of generic scenarios comprised of combinations of events and facts about players involved in those events. This problem requires more than a simple database querying capability because of several sources of ambiguity and uncertainty within the process. There may be errors, noise, or missing information within the data. The same or similar situations may

[1] Here and throughout the remainder of the paper, we use the term "analyst" as a shorthand to describe the intended end-user of LAW. In practice, LAW is more broadly intended for any intelligence professional, from low-level analysts to high-level managers in the intelligence community.

be represented in multiple different ways. And most importantly, the analyst may only be able to describe the situation of interest at a high level or with a limited amount of precision.

The model of patterns LAW uses for this problem of approximate matching is to have each pattern represent two things: (1) a *prototype* of the situation of interest, and (2) allowable *deviations* from the prototype, along with the impact these deviations have on the assessed quality of the match. To fill this model and to meet the additional requirement of understandability, we have chosen a pattern representation based on graphs, and a match metric based on *graph edit distance*.

Pattern Representation

A LAW pattern consists of two parts: a graph, representing the prototype situation described above; and a collection of edit distance parameters, representing the allowable deviations described above. We describe the former here, and describe the latter below under the Pattern Comparison Metric heading.

The graph portion of the pattern representation (called the *pattern graph*) is a collection of typed vertices (nodes), and a collection of labeled edges (links) relating the vertices. Each node in the graph is a *generic concept* (Sowa 1984) of a type, or a literal. The types are organized in an ontology; to access the ontology in LAW, we are using OCELOT (Paley, Lowrance, & Karp 1997), an OKBC-compliant (Chaudhri *et al.* 1998) knowledge representation system. Labels on edges are also typed with the types organized in the ontology. Specific instances can be approximated in the pattern using literals. For example, the person Ruth Ann Steinhagen can be represented by attaching a PERSON node to a node containing the string "Ruth Ann Steinhagen" via a NAME relation.[2]

Our design goal for the pattern graph representation is to give to the analyst a representational capability that is powerful, but still understandable to a lay-user and reasonably efficient to match. In particular, we want a pattern language that stops well short of the representational power and inferential capabilities of first-order logic or conceptual graphs (Sowa 1984), but still goes beyond the capabilities of simple flat typed graphs. We have recently extended the design of the pattern graph representation to include notions of hierarchy, disjunction, and cardinality.[3] For example, Figure 1 shows a hierarchical pattern representing a murder-for-hire. The pattern hierarchically includes two subpatterns, one representing a phone call (phoneCallPattern), and one representing a meeting (meetingPattern). In this figure, only the interface nodes of the subpatterns are shown; the internal structure is hidden. We are currently upgrading the pattern

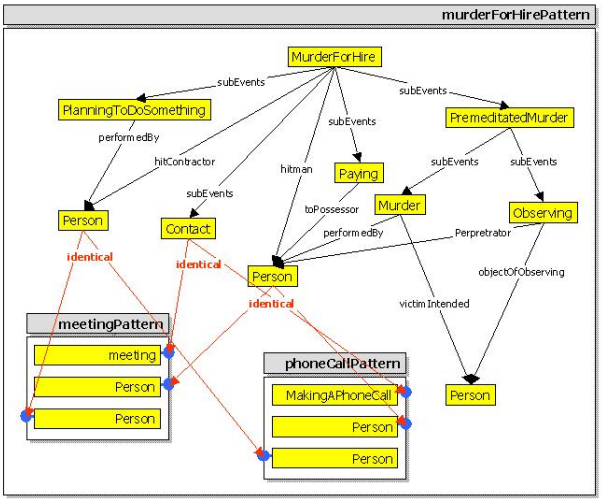

Figure 1: Graphical depiction of a Murder-For-Hire pattern

comparison metric and the matching algorithm to handle the more advanced pattern graph representation. LAW does currently have an ability to match graphs hierarchically, but it is primitive. The less sophisticated, flat graph the current implementation uses to represent murder-for-hire is shown in Figure 3.

Pattern Comparison Metric

The term "graph edit distance" covers a class of metrics that measure the degree of match between two graphs. Variants have been studied theoretically (Bunke 1997; Bunke & Shearer 1998) as well as applied in domains as diverse as image understanding (Shapiro & Haralick 1981) and reasoning by analogy (Wolverton 1994). In its simplest form, the graph edit distance between two labeled graphs G_1 and G_2 is the smallest number of editing operations it would take to transform G_1 into G_2. Allowable editing operations are node addition, node deletion, edge addition, edge deletion, node label replacement (i.e., changing the label attached to a node from one term to another), and edge label replacement. This simple model can be extended by adding costs to the various editing operations, perhaps as a function of the labels on nodes or edges, and measuring the edit distance between two graphs as the minimum cost of a sequence of operations that converts one into the other.

LAW uses the more complex model of associating costs with operations. The current LAW model uses only three of the six aforementioned editing operations: node deletion, edge deletion, and node replacement.[4] Each node and edge in a LAW pattern graph has an associated cost for deleting

[2]This encoding represents *any* person named Ruth Ann Steinhagen, which is often sufficient for the purposes of pattern matching. If the user needs to identify the person of interest more narrowly, additional qualifiers (e.g., birthdate) can be used.

[3]By cardinality, we mean specifying information about the number of links, nodes, or subgraphs, e.g. "three or more meetings."

[4]Node addition and edge addition are not relevant in pattern matching (unlike, for example, analogical reasoning), because of the asymmetry between pattern and data: you aren't trying to make the pattern look like the entire data set, only a small portion of it And while edge replacement could be a useful construct in pattern matching, we have not yet found a need for it in practice in our use of the system.

it (see below). For node label replacement, LAW uses the ontological distance between the types of the pattern node and the mapped data node.

The collection of edit distance parameters contained within a LAW pattern specify the allowable deviations from the prototype that will still be considered a valid match, and the cost that various deviations have on the overall quality of the match. These parameters control the calculation of the edit distance between the pattern and the data. They include:

- a *deletion cost* on each node and link in the pattern. Each of these can be a number, which roughly reflects the node's or link's level of importance in the pattern. They can also be set to a symbol representing infinite cost, which indicates that the node or link must be matched by a node or link in the data.
- a *maximum ontological distance* on each node in the pattern. This specifies the allowable distance between the type of a pattern node and the type of a node in the data that matches it. Setting this to 0 indicates that the pattern node must be matched exactly, e.g,, a PHONE-CALL node in the pattern can only match a PHONE-CALL node in the data. Setting it to a number greater than 0 indicates that the node can be matched to a node of another type, e.g., a PHONE-CALL node in the pattern can be match other subtypes of COMMUNICATION.
- an *ontological distance multiplier*, which specifies the cost of mapping a node of a given type in the pattern to a node of another type in the data. This factor specifies how much penalty the match will pay, for example, for matching a PHONE-CALL node in the pattern to an EMAIL node in the data.
- a *maximum total edit distance* for allowable matches. No matches that are above this threshold will be returned, and any partial matches that exceed this threshold will be pruned from the system's search.
- the *maximum number of matches* for the system to return.

Matching Algorithm

LAW's current approach to finding the closest matches to the pattern in the data is based on A* search (Hart, Nilsson, & Raphael 1968). A state in the search is a partial match—a mapping between a subset of the pattern nodes and data nodes, a mapping between a subset of the pattern links and data links, a set of unmapped nodes, a set of unmapped links, and a cost of the mappings so far. The cost is the sum of the delete costs of the unmapped nodes and link, and replacement cost of the node and link mappings, as described above.

LAW generates start states for the search by selecting the node in the pattern with the fewest legal mappings in the data and creating a partial match for those mappings. It expands a partial match by selecting an unexplored node mapping $(PatternNode, DataNode)$ and generating new mappings for each link adjacent to $PatternNode$ to every mappable link adjacent to $DataNode$. When a pair of links is mapped, the nodes on the other ends of those links are mapped as well.

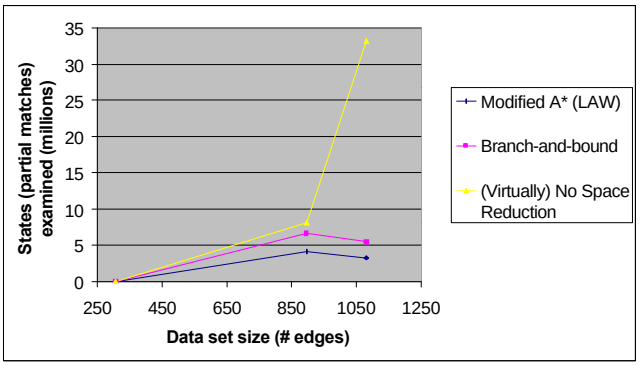

Figure 2: Effects of heuristics in pattern search

The search selects as the next state to expand the one with the minimum worst-case cost—i.e., the cost of the mappings so far plus the cost of deleting all unexplored nodes and links in the pattern. Since the cost of deleting all unexplored nodes is guaranteed to be an upper bound on the eventual cost of any extension to that partial mapping, this selection heuristic meets A*'s admissibility criterion, and the search is guaranteed to find the lowest-cost solution. The search prunes any partial match that cannot possibly have a lower cost than the best n matches found so far, where n is the maximum number of matches the user wants to see, as well as any that cannot possibly have a lower cost than the pattern's maximum allowable cost. In addition, at the end of the process LAW prunes any mappings that are *subsumed* by other discovered mappings. A mapping A subsumes a mapping B if $MappedEntities(A) \subseteq MappedEntities(B)$, i.e., if they differ only in that B has more node and link deletions than A.

The search process is designed to find a good set of pattern matches quickly, and then use those existing matches to prune the remainder of the search. One key asset of the approach is that it is an *anytime* algorithm: at any point during the process the algorithm can return the set of matches it has found already, and that set of matches will monotonically improve as the process continues.

The performance of the algorithm has not yet been rigorously evaluated, but our preliminary experience using the system on evaluation data sets suggests the pattern matcher's speed is tolerable, if not yet scalable to the size data sets to which it will eventually be applied. The pattern matcher completed the DARPA EELD 2002 evaluation problem—involving matching patterns of 20-30 nodes against 28 fabricated data sets of varying size—in a few hours, which was comparable in speed to other link discovery tools evaluated.

Figure 2 shows the benefit of different aspects of the approach. The top line shows the amount of effort it takes to match data sets of varying size with relatively little search control. The middle line shows the performance of branch-and-bound—i.e., pruning the search as described above, but selecting the next state to explore in depth-first order rather than a heuristic evaluation function. The bottom line adds the evaluation function selection to convert the branch-and-

bound approach to A*. One thing the graph demonstrates is that the match time is not completely determined by data set size; both the heuristic approaches were slower in the mid-size data set than they were in the largest one. The match time will be dependent on a large number of factors in addition to data set size, including the size of the pattern, the number of good partial matches in the data, and the pattern-defined maximum allowable cost of a match.

Example and Interface

This section has two purposes. First, it provides an example of the user's interaction with LAW—and especially the LAW pattern matcher described in the previous section—through screen shots of the system. Second, it discusses the design of LAW's user interface, which is based on the common pattern framework and web services architecture described in the next section.

Interface Design

LAW's interface runs primarily through the web. The web-based implementation offers two main advantages. First, any connected computer with a browser can access it without the need for an error prone installation cycle. Second, it cuts to zero the time from release to deployment, and it allows concerns of the users to be answered, implemented, and delivered much more quickly. On the other hand, the disadvantage of a browser-based interface is that there is only so much you can do using HTML and JavaScript. Highly interactive HTML interfaces are either kludgy and inefficient or impossibly slow. For that reason, certain highly interactive pieces of the LAW interface—the pattern editor in particular—are implemented as local applications that connect to the original server using web services.

Given the pattern matching task described above, a user interface must provide a way to make sense of the structure and amount of the information available. It must also provide a pattern editor, that permits the creation of patterns in terms of the data. The most important component, the pattern matcher has limited user interface. When matches are extracted, they must be shown in a drillable way, that is consistent both with the views of the data and the pattern construction.

Visualization Visualization in LAW can be divided into three areas: patterns, the data to which the patterns are matched, and the ontological concepts of which the pattern and the data are composed.

As described earlier, patterns contain two elements: a graph that represents the prototype situation, and the edit distance parameters that represent the allowable deviations from the prototype. LAW's presentation of patterns displays both elements. Figure 3 shows LAW's display of a pattern. The display shows the objects and relations of the pattern (the nodes and links) as well as the edit distance parameters (encoded in colors of the nodes and links).

Ontology management is a well-studied problem (Paley, Lowrance, & Karp 1997). Since ontologies are not a primary focus of LAW, its ontology exploration and editing capabilities are limited. Figure 4 shows the user exploring

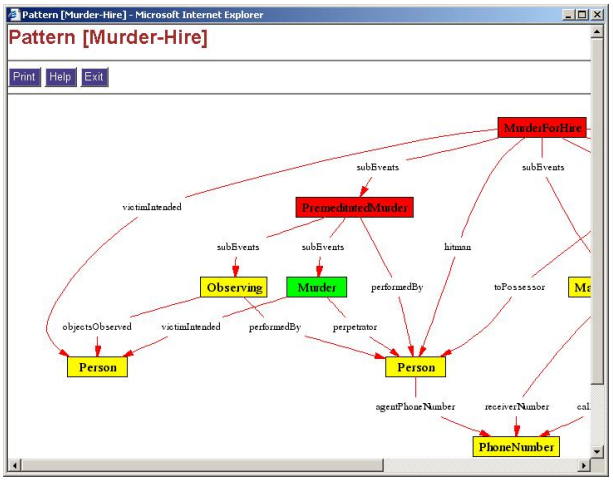

Figure 3: LAW's display of the Murder-for-Hire pattern

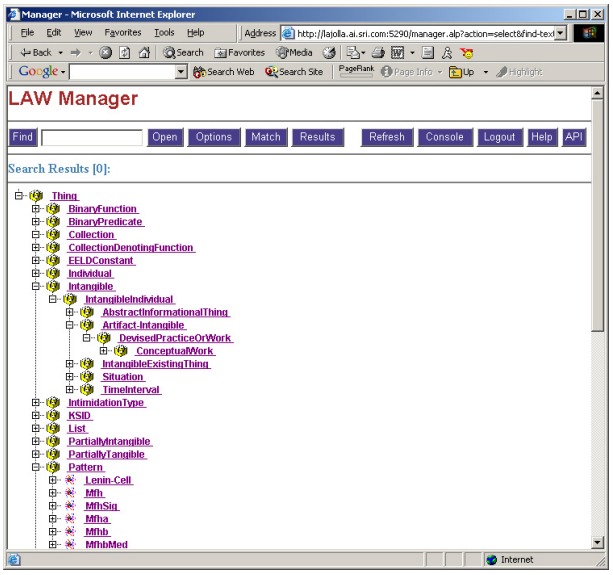

Figure 4: Ontology browsing

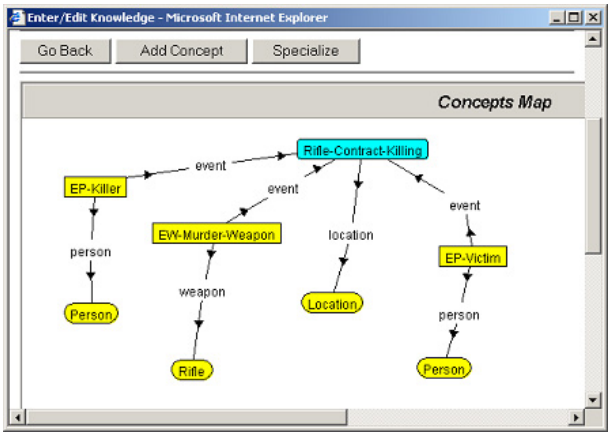

Figure 5: SHAKEN interface

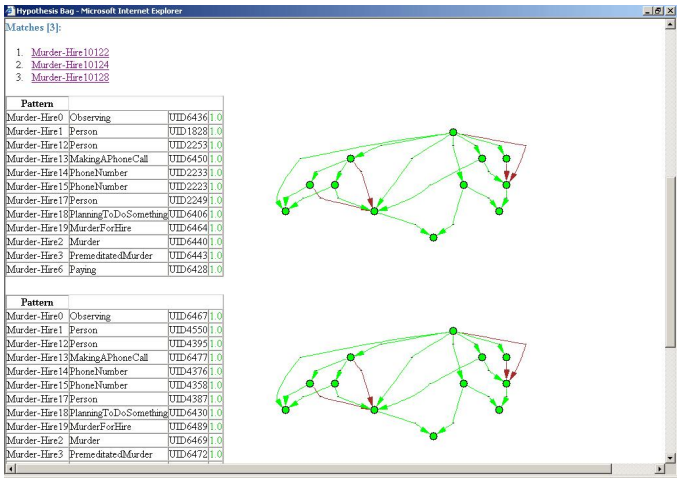

Figure 6: Results list page

the ontology via LAW's hierarchical browser. Although the range of things you can do to the ontology is restricted, our tool can import DAML (Hendler & McGuinness 2000) or CycL (Cycorp 2003) ontologies, so modification of these is permitted by any tool that is able to export in those formats.

The last area, visualization of primary data, has not played an important role in our work on this project up to this point.

Pattern Editing LAW's development has been guided by our view that pattern matching should be an interactive task, where patterns will be continuously refined during an iterative matching process. Some patterns will even have to be created either from scratch (i.e. directly from concepts in the ontology) or from other patterns.

The interactive nature of the pattern matching process requires that the user be able to revise his thinking quickly, which in turn requires the ability to make fast modifications to the graph. A browser is ill-suited for this type of interaction. In the current version of LAW, we are using a knowledge acquisition tool called Shaken (Thomere *et al.* 2002) for the construction of patterns. Figure 5 shows the user editing a pattern in Shaken.

The requirements for a pattern editor include: ability to edit and/or incorporate existing patterns, coming from other sources, ability to build patterns from concepts in the ontology, handling of small or very big patterns (possibly thousands of nodes). The pattern editor is basically a graph editor, but with a constrained set of possible operations. It is also component-based, which means that patterns can be made of other patterns. The main operations that can be performed on a pattern are: the addition of a node, the connection of two nodes and the merging of two nodes.

Pattern Match Visualization The final piece of the interface is the pattern match (result) visualization. Just as we emphasized how important it is that we can observe the data and the patterns under the same framework, the same is true for the results. This is important so that the relationship between the pattern and the results instantiating it is self evident. If a mental mapping between one and the other is difficult, then the ability to modify the pattern as we look at the results is diminished.

When a request for matches is initiated, the server will asynchronously start the search and inform the user of it. When the results are available, the ones that meet the user's criteria are presented, ordered best to worst. The images of the results are small replicas of the graph used to describe the pattern. The nodes of the result image are colored to display the degree of accuracy obtained in the match. Figure 6 shows the results of the match of Figure 3's pattern.

Although not implemented currently, the idea is that these matches can be reentered into the system as data, so that they can be revisited later on. This scenario makes sense in the case of streaming data, where the system does not yet know if more evidence is going to become available. There must be a big portion of the system devoted to what we call "hypothesis management"—that is, storing, revising and comparing past results. The current system offers a good explanation of why a match coincides with the pattern offered, by assigning scores to all the nodes and relationships.

Architecture

While the LAW pattern matcher described in the last two sections provides a powerful and flexible mechanism for finding partial matches to patterns, it cannot possibly cover all the pattern matching criteria an analyst could have. There are many tools under development, general and specialized, that support the many complementary data exploration requirements of the intelligence community. In a given search session, an analyst may want to combine results from an approximate match to a general scenario (as retrieved, e.g., by the LAW matcher just described), a general probabilistic reasoning engine (e.g., (Taskar, Abbeel, & Koller 2002)), and a specialized tool for determining group membership (e.g., (Kubica *et al.* 2002)).

We have developed and implemented an architecture for LAW that supports integration of multiple pattern matching tools. It includes a language for sharing patterns, and a web services architecture that makes integration straightforward.

The LAW user interface allows a user to interact with these tools through a common framework. Right now, the user must select and task the tools manually, but our aim is to have the system provide support for tasking of tools through LAW's Control component.

High Level Architecture

LAW is architected as a web application, made up of a number of web service components. At the heart of LAW is a knowledge base server, with an in-built web server. Data for this knowledge base server can reside in either a database or a flat file. The client side of LAW is ephemerally generated web pages, encoded in HTML, Javascript and embedded Java applets, with the client being any standard modern web browser.

This thin client, web server architecture had previously been used for a successfully fielded collaborative structured argumentation application, SEAS (Lowrance, Harrison, & Rodriguez 2001). The adoption of a web browser as the client, removed the need to install and maintain client software in end-user organizations. For LAW, we decide to enhance this architecture, using a web service model. Our driver for this was that partner companies were simultaneously developing applications, which we wanted to make available to the LAW user through its GUI. Likewise, several of the LAW components were of general value to external partners, outside of the LAW framework. Given that we were not able to state at the outset all the different ways in which we and other partners would integrate the various components into larger systems, web services offered an ideal way of providing a lightweight, flexible means of integration. The idea was that during the development of the various components, and as our understanding of the domain was refined, we would be able to experiment with different configurations of web services to provide different, complementary, end user solutions.

From a user's perspective, they are unaware of the configuration/location of the underlying services, such as pattern matching. Instead they are presented with an interface, which can hide all the integration details. For instance, LAW is being designed so that its underlying control structure can break down pattern match tasks into smaller tasks, send these match tasks to different, appropriate match services, and then merge the results back into a whole. All this can happen totally automatically, without the intervention of a user. The Future Work Section discusses this in more detail.

Figure 7 depicts the current LAW architecture. In the current architecture, LAW can be viewed as having three internal components:

- the control component that is responsible for coordinating the tasks between the user input (via UI) and the other (internal and external) components.
- the UI component, that is the user interfaces that are presented to a user to request pattern matches and to view the results of matches
- the Pattern Match component

LAW itself contains a web server, which is used to communicate between the different internal components, as well as external components, with the messages being encoded in XML, conforming to defined schema (PatternML, HypothesisML and ControlML). External components are all intended to be tasked using SOAP, although currently some components are still tasked through custom interfaces.

The LAW server sits on top of SRI's Ocelot knowledge base server, with communication between LAW and the server via OKBC. This knowledge base server contains the domain ontologies together with the evidence data sets that are to be matched against.

The LAW pattern editor is currently a separate application, the Shaken editor (Thomere *et al.* 2002). Pattern sdefined using this editor are stored in LAW's Ocelot knowledge base server

SOAP

We chose to adopt SOAP (Simple Object Access Protocol - www.w3.org/TR/SOAP) as the standard communication protocol between LAW's internal and external web services. SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol, which is neutral about the actual transport mechanism. This means that HTTP, SMTP, raw TCP, an instant messaging protocol like Jabber, etc., could potentially be used for message transport. For LAW we decided to adopt HTTP as the communication protocol. LAW uses SOAP to enable remote procedure calling (RPC), where an RPC call maps naturally to an HTTP request and an RPC response maps to an HTTP response. SOAP was also attractive from an architectural standpoint in that it does not require that a message be sent from a client to a server in a single "hop". The SOAP specification defines the notion of intermediaries, nodes that a message passes through on its way to its final destination. Using intermediaries, you can "virtualize" physical network topology so that messages can be sent to web services using whatever path and whatever combination of transport protocols is most appropriate. This last capability facilitates our vision for the LAW architecture as a federated collection of services that communicate via the Internet to coordinate their search for pattern matches over distributed data sources, in service of a community of intelligence analysts

XML Schema

To allow communication between various components internal to LAW and external component web services, we needed to define an XML schema for the content of the SOAP messages, so that all components could understand the syntax of the content, and map the content to their internal representations. We adopted a layered approach to schema design, where the core is a schema to describe template domain patterns (we called this schema PatternML). PatternML was designed to have two main uses: as an interchange language between pattern editors/visualizers; and as an input format to pattern matching components. Patterns defined using PatternML made reference to existing ontologies for domain concepts. Layered on top of PatternML was the HypothesisML schema, which was designed to be a common output format from pattern match components

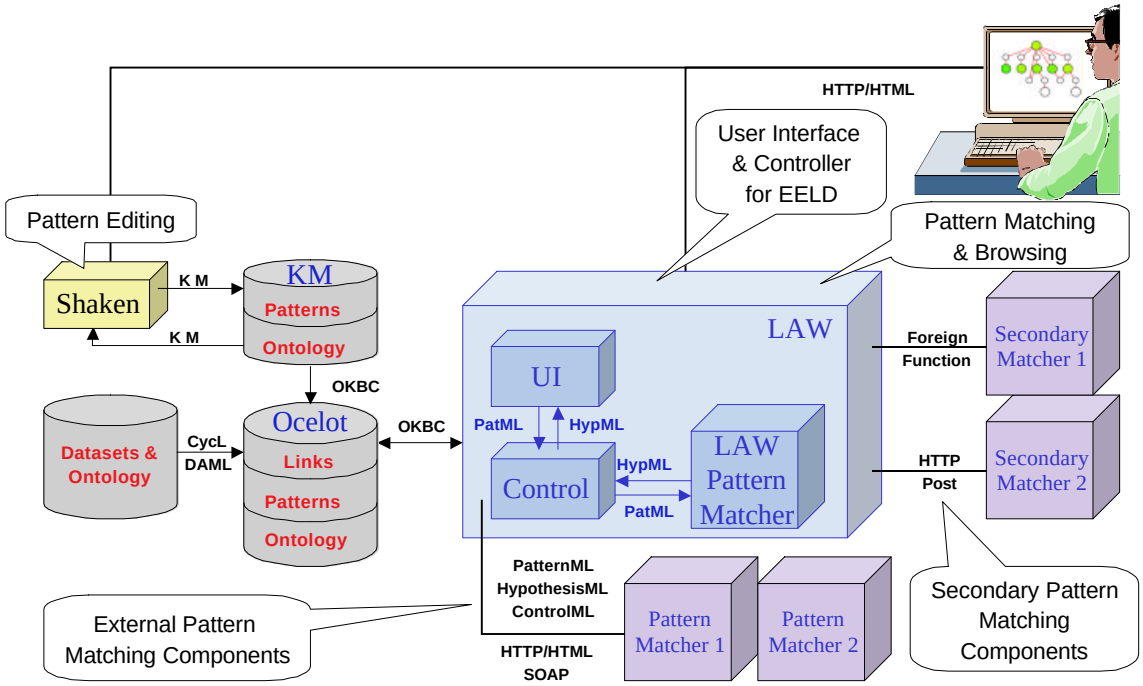

Figure 7: LAW Architecture

and as an input format for hypothesis visualizer or hypothesis management components. HypothesisML describes the match between a PatternML patterns and a particular evidence data set. Layered on top of both of these schemas was the ControlML schema, which was designed to describe intended task control information. ControlML allows one component to describe what it wishes other components to do, together with various control parameters, and allows PatternML or HypothesisML as content (to describe pattern templates or hypothesized pattern match(es), respectively).

Control

An analyst can use LAW for a variety of purposes—for example, to develop, test and refine a new pattern, to make a one-off request to match a pattern over a specified data source, or to monitor data sources at a regular interval for a group of predefined patterns. These purposes, and more, are supported by an underlying control methodology, based on *dynamic process management*. The LAW controller combines the highly reactive techniques employed in intelligent process control domains with the more accountable and systematic approach to business management provided by the field of workflow management systems.

The implementation uses the ACT formalism, (Myers 1993), to encode the procedural knowledge necessary to capture the flows of control. These can be viewed graphically so that the work processes are accessible to the user. This feature will prove useful in the future when we encode search strategies in the same way. Each ACT represents a specific flow of control in a given context. The SRI Procedural Reasoning System (PRS), (Georgeff & Ingrand 1989), forms the basis for a hierarchical, reactive control system to manage requests from the user interface, e.g. one-off, repetitive, or periodic requests. Using a workflow management model of tasking,(Berry & Drabble 1999), PRS also coordinates the tasking and monitoring of available pattern matching tools including the LAW matcher. Experiments have included the dynamic tasking of the LAW matcher and two other pattern matching tools via web interfaces.

Future Work

The current version of LAW represents a step toward the kind of user-centric, approximate pattern matching tools required by the intelligence community. But there are many important issues that remain to be tackled. Here are a few of the needs that we consider the most pressing.

Expanded Pattern Representation and Matching

As discussed above, we have designed extensions to LAW's pattern representation language that incorporate hierarchy, disjunction, and cardinality within patterns. One current area of research and development is adapting LAW's pattern comparison metric and match algorithm to handle this expanded representation. Particular challenges include producing an extended graph edit distance metric that sensibly deals with cardinality, and exploiting the hierarchical structure of patterns to find good matches more quickly than for flat patterns.

Scalability

While LAW's current pattern matching approach is efficient enough to assist an analyst in investigating relatively small data sets, a great deal of work needs to be done to support approximate pattern matching on a very large amount of data. Approaches to scale LAW to large data sets will include: incorporating relational database storage and retrieval technology into the system, domain-dependent control information that restricts the data under consideration to a small subset of the entire data space, and distributing a pattern match to multiple processors and/or tools via the Control component.

Pattern-Specified Control

This is another approach to improve scalability that deserves separate mention. Currently, LAW allows the analyst to specify the degree of importance of particular nodes and links in the pattern, through the use of the pattern's edit distance parameters. But we would like to give the analyst even more ability to bring his expertise to bear in directing the search for matches. In particular, we will investigate ways of giving him procedural control over the pattern match: e.g., which node(s) in the pattern to attempt to match first, and in which data source(s) to look.

Automated Tasking of Pattern Matchers

As described above, LAW's architecture provides a mechanism for the integration of multiple pattern matching tools, and gives the user the ability to select a tool for a particular pattern matching task. In order to fully and flexibly support a wide variety of pattern matching needs, however, the system should have some ability to automatically task the tools based on their capabilities. This tasking requires two main technical components; (1) the modelling and representation of capability, and (2) the mechanisms to support web-based tasking, described in a previous section.

The approach we envision will enable semantic tasking by modelling the capabilities and properties of a particular tool using a methodology developed for workflow management system SWIM (Berry & Drabble 1999). The type of characteristics to be modelled includes the service itself, data over which the service is applicable, strengths and weaknesses according to pattern structure and pattern semantics, performance, and response time.

Acknowledgements

This research was supported by the Defense Advanced Research Projects Agency (DARPA) under Air Force Research Laboratory (AFRL) contract number F30602-00-C-0193.

References

Berry, P., and Drabble, B. 1999. Swim: An ai-based system for workflow enabled reactive control. In *In proceedings of the IJCAI Workshop on Workflow and Process Management held as part of IJCAI-99*.

Bunke, H., and Shearer, K. 1998. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters* 19:255–259.

Bunke, H. 1997. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters* 18:689–694.

Chaudhri, V. K.; Farquhar, A.; Fikes, R.; Karp, P. D.; and Rice, J. P. 1998. OKBC: A programmatic foundation for knowledge base interoperability. In *Proceedings of the AAAI-98*.

Cycorp. 2003. Ontological engineer's handbook. http://www.opencyc.org.

Georgeff, M. P., and Ingrand, F. F. 1989. Decision making in an embedded reasoning system. In *Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* SSC-4(2):100–107.

Hendler, J., and McGuinness, D. L. 2000. The DARPA agent markup language. *IEEE Intelligent Systems* 67–73.

Kubica, J.; Moore, A.; Schneider, J.; and Yang, Y. 2002. Stochastic link and group detection. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*.

Lowrance, J. D.; Harrison, I. W.; and Rodriguez, A. C. 2001. Capturing analytic thought. *Proceeding of the First International Conference on Knowledge Capture* 84–91.

Myers, K. L. 1993. *The ACT Editor User's Guide*. Artificial Intelligence Center, SRI International, Menlo Park, CA.

Paley, S. M.; Lowrance, J. D.; and Karp, P. D. 1997. A generic knowledge-base browser and editor. In *Proceedings of the 1997 National Conference on Artificial Intelligence*.

Shapiro, L., and Haralick, R. 1981. Structural descriptions and inexact matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 3:504–519.

Sowa, J. F. 1984. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley.

Taskar, B.; Abbeel, P.; and Koller, D. 2002. Discriminative probabilistic models for relational data. In *Eighteenth Conference on Uncertainty in Artificial Intelligence*.

Thomere, J.; Rodriguez, A.; Chaudhri, V.; Mishra, S.; Eriksen, M.; Clark, P.; Barker, K.; and Porter, B. 2002. A web-based ontology browsing and editing system. In *Conference on Innovative Applications of Artificial Intelligence*. AAAI.

Wolverton, M. 1994. *Retrieving Semantically Distant Analogies*. Ph.D. Dissertation, Stanford University.