An Architecture for Information Agents

Donald P McKay, Jon Pastor and Robin McEntire
Loral Defense Systems

Tim Finin
Computer Science and Electrical Engineering
University of Maryland - Baltimore County

Abstract

Information agents include a significant class of applications which mediate information structures of domain objects to instance representations in a storage manager. Over the past several years, we have been experimenting with an information agent architecture in the context of the ARPI. Our information agent architecture uses the Knowledge Query and Manipulation Language (KQML) to implement access the knowledge services of such an information agent. The information agent itself, which we call the Loom Interface Module (LIM), uses knowledge structures to represent domain objects and contains an explicit mapping of knowledge structures to representations in an external storage manager, a relational database management system. We have developed several performance metrics and features for information agents constructed using this architecture. We described several key component algorithms and performance measurements We have developed the performance metrics, analysis and examples as a part of ARPI TIEs, introduction into the Common Prototyping Environment and, most importantly, under collaboration with the SIMS project at USC ISI and with the CoBASE project at UCLA.

Introduction

Knowledge-based systems can provide a key information processing aid to operational planning, scheduling and monitoring of operations. Specifically, these systems can provide key information support for current deficiencies in crisis action planning for transportation logistics. Requirements for these systems include the ability to access, manipulate, and modify the information stored in existing databases, and, a high level of collaborative and cooperative processing with the other planning agents including people and software components. Within the ARPA/Rome Lab Planning Initiative (ARPI), Loral Defense Systems, in collaboration with USC ISI and UCLA, developed an intelligent information services architecture which integrates cooperative user interaction and information location via domain/user-oriented object representations. This effort, involving participants and software components developed by Loral Defense Systems, USC ISI and UCLA, demonstrated an experimental prototype operating in real-time over the internet capable of providing information satisfying user requests making transparent to the user 1) query relaxation and reformulation despite over-specific queries and lack of data, 2) location and selection of information sources based upon multiple selection criteria, 3) transformation of low-level data source information from databases into domain and user relevant information structures, and 4) the query language utilized. Internal communications over the internet were implemented using KQML, the Knowledge Query and Manipulation Language, an ARPA-sponsored emerging language and protocol for information exchange.

In this paper, we describe technology components to support persistent storage and retrieval of plans and other military transportation relevant entities. This includes the integration of knowledge-based (KB) representation and reasoning systems with standard database (DB) management systems and the development of new standards for interface languages between knowledge-based systems and other software components including knowledge-based systems themselves. The integration of knowledge bases and databases is accomplished by the Loom Interface Module (LIM). LIM allows Loom (MacGregor & Bates 1987) applications to reason efficiently over a large collection of data from a database by utilizing the efficient computational capabilities of a database management system and by avoiding the need to create regular Loom objects to represent intermediate data. In order to enhance the integration of multiple knowledge-based systems, Loral Defense Systems and UMBC are designing and prototyping a new high-level protocol for conveying knowledge between systems. This protocol,

[1] This work was supported by Rome Laboratory and the Advanced Research Projects Agency under USAF contracts F30602-93-C-0177 and F30602-93-C-0028. The views and conclusions are the authors' and should not be interpreted as the official opinion or conclusions of the US Government, the USAF, Rome Laboratory, or ARPA.

KQML (Knowledge Query and Manipulation Language), is being developed in conjunction with a number of university and industry laboratories under the ARPA Intelligent Information Integration program and the Knowledge Sharing Initiative.

These two components, when integrated with other intelligent information system components being developed at USC ISI (SIMS) and at UCLA (CoBASE), provide intelligent access to distributed information sources in a fault tolerant and cooperative manner supporting military planners. The SIMS (Arens 1992) and CoBASE (Chu & Chen 1994) systems are described elsewhere.

ARPI Information Agent

This section describes the basic architecture of an Information Agent --- a knowledge server or source capable of handling all requests for information in a given domain, in this case, the transportation logistics planning domain. We have constructed an Information Agent prototype based on the Loom Interface Module (LIM). Using LIM, we have constructed an Information Agent (see Figure 1) which mediates between knowledge structures defined for use by intelligent system components and database structures. This information agent responds to queries and other commands which operate upon knowledge structures and translates them to the appropriate target system, e.g., SQL queries and data manipulation commands. This information agent has been used to support experimental representations of transportation assets (e.g., planes and ships), geographical locations (e.g., airports and seaports) as well as transportation relevant information about forces and transportation schedules. This LIM information agent is used in conjunction with the CoBASE and SIMS systems described elsewhere to provide a flexible and distributed cooperative intelligent information agent for transportation data which can be accessed at each of these interface points. If only mediation to shared representations is desired, the LIM information agent can be accessed directly; if information access planning is required the SIMS agent can be accessed; finally, if cooperative processing is desired, CoBASE can be used as the point of contact. All three systems can be accessed independently depending on the desired functionality. The Knowledge Query and Manipulation Language is used to support this level of communication transparency.

We have built an Information Agent prototype which involved the integration of the three knowledge-base/data-base components: LIM, SIMS, and CoBASE and focused upon the data and information collected for the transportation logistics domain. The prototype also tested the robustness of its three component systems in a realistic

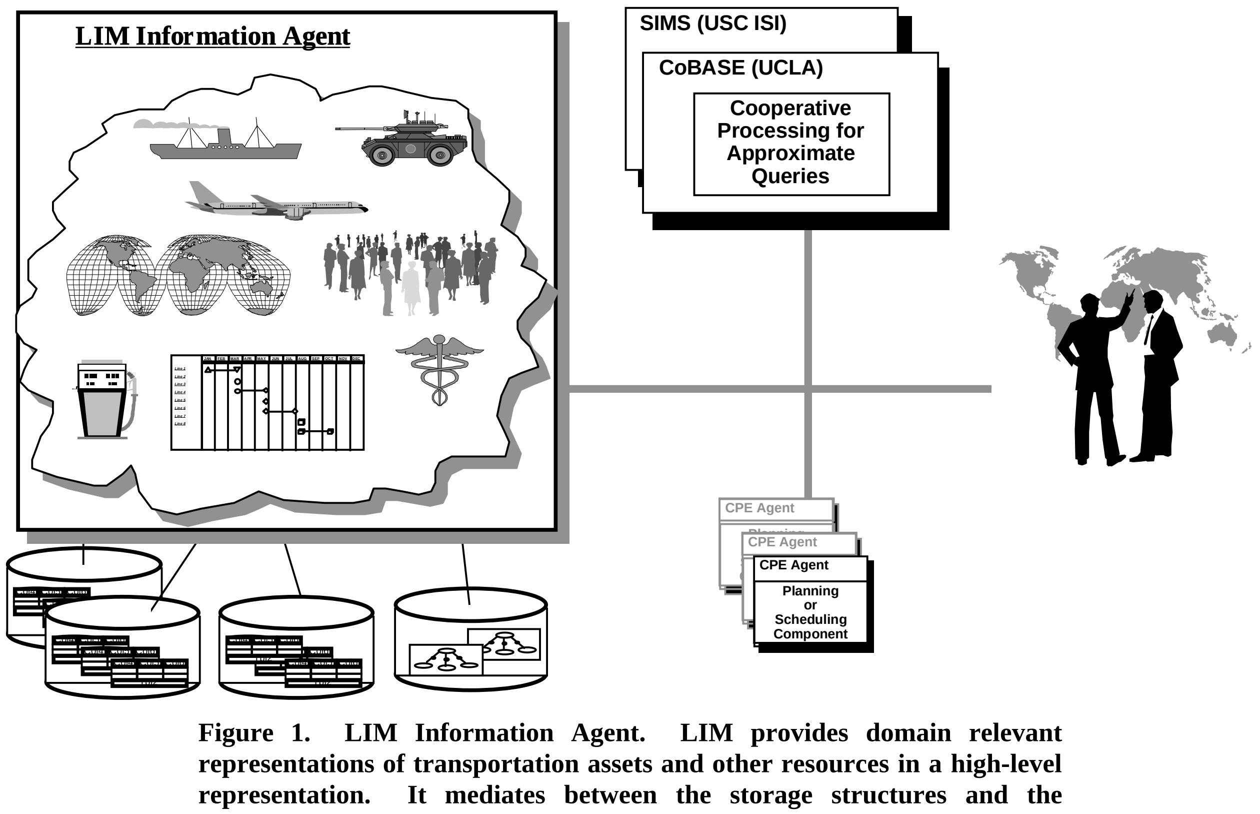
LIM Information Agent
SIMS (USC ISI)
CoBASE (UCLA)
Cooperative Processing for Approximate Queries
CPE Agent
CPE Agent
CPE Agent
Planning or Scheduling Component

Figure 1. LIM Information Agent. LIM provides domain relevant representations of transportation assets and other resources in a high-level representation. It mediates between the storage structures and the representations used by other intelligent agents.

information environment. Performance of tasks in the transportation planning domain typically requires access to data stored in a multiplicity of databases, by people (or computer systems) unfamiliar with their specific structure and contents. It is thus necessary to provide for the possibility of retrieving required data using a uniform language, independently of where the data is actually located and how complicated the actual process of retrieving it may be.

The Information Agent architecture currently address separate aspects of this problem. This prototype united them into one system that:

- accepted a query in an extension of the Loom language,
- relaxed the query, if appropriate, to enable retrieval of additional information of relevance to the user,
- planned a series of queries to databases and data manipulations that
- brought about the retrieval and/or computation of the
- requested data, and finally
- execute the plan, issuing the necessary queries to the appropriate databases, and returned the resulting data to the user

LIM, SIMS, and CoBase have been combined in various ways, including both a single Common Lisp program which shared one Loom model of the application domain and the databases as well as a distributed Information Agent architecture in which the LIM Information Agent, acting as a server, was at a remote site. Queries were submitted in the Loom language, extended by the approximation operators supported by the CoBASE system. CoBASE translated the user's query into one in the standard Loom language. SIMS broke down the resulting query into a series of LIM queries (again in the Loom language), each restricted to a single databases. The databases were accessed over a network, using the LIM database interface.

KQML Agent Communication Language

This section provides a brief overview of the agent communication language used in the Information Agent architecture. Many computer systems are structured as collections of independent processes, frequently distributed across multiple hosts linked by a network. Database processes, real-time processes and distributed AI systems are a few examples. Furthermore, in modern network systems, it should be possible to build new programs by extending existing systems; a new small process should be conveniently linkable to existing information sources and tools such as filters or rule based systems.

One type of program that would thrive in such an environment is a mediator (Wiederhold 1992), or information agent in this paper. Mediators are processes which situate themselves between "provider" processes and "consumer" processes and perform services on the raw information such as providing standardized interfaces; integrating information from several sources; translating queries or replies. Mediators are becoming increasingly important as they are commonly proposed as an effective method for integrating new information systems with inflexible legacy systems.

Standards and intercommunication approaches such as CORBA, ILU, OpenDoc, OLE, etc., are efforts that are often promulgated as solutions to the agent communication problem. Driving such work is the difficulty of running applications in dynamic, distributed environments. The primary concern of these technologies is to ensure that applications can exchange data structures and invoke remote methods across disparate platforms. Although the results of such standards efforts will be useful in the development of software agents, they do not provide complete answers to the problems of agent communication. After all, software agents are more than collections of data structures and methods on them. Thus, these standards and protocols are best viewed as a substrate on which agent languages might be built.

KQML is a language and a protocol that supports this type of agent communication specifically for knowledge-based systems or information agents. It was developed by the ARPA supported Knowledge Sharing Initiative (Neches et al. 1991, Patil et al. 1992) and separately implemented by several research groups. It has been successfully used to implement a variety of information systems using different software architectures.

KQML is a layered agent communication language (Finin et al. 1994; Finin et al. 1995; Mayfield et al. 1996). The KQML language can be viewed as being divided into two layers: the content layer, and the message layer or the communication layer. The content layer is the actual content of the message, in the agent's representation language; in the Information Agent described in this paper the content language was an extension of the Loom language developed under the Planning Initiative. KQML can carry any representation language, including languages expressed as ASCII strings and those expressed using a binary notation. All of the KQML implementations ignore the content portion of the message except to the extent that they need to determine where it ends.

The communication level encodes a set of features to the message which describe the lower level communication parameters, such as the identity of the sender and recipient, and a unique identifier associated with the communication. It also determines the kinds of interactions one can have with a KQML-speaking agent. The primary function of the communication layer is to identify the protocol to be used to deliver the message and to supply a speech act or performative which the sender attaches to the content. The performative signifies that the content is an assertion, a query, a command, or any of a set of known performatives. Because the content is opaque to KQML, this layer also includes optional features which describe the content, e.g., its language.

Conceptually, a KQML message consists of a performative, its associated arguments which include the real content of the message, and a set of optional arguments which describe the content in a manner which is independent of the syntax of the content language. For example, a message representing a query about the location of a particular airport might be encoded as:

```
(ask-one :content (GEOLOC LAX (?long
    ?lat)) :ontology GEO-MODEL3)
```

In this message, the KQML performative is ask-one, the content is (geoloc lax (?long ?lat)) and the assumed ontology is identified by the token :geo-model3. The same general query could be conveyed using standard Prolog as the content language in a form that requests the set of all answers as:

```
(ask-all    :content
            "geoloc(lax,[Long,Lat])"
            :language standard_prolog
            :ontology GEO-MODEL3)
```

Loom Interface Module

LIM acts as an intermediary between a Loom application and one or more DBs. The inter-relationships among the various components of the overall system are illustrated in Figure 2. LIM uses the DB schema, building a Loom representation of the schema based on this information. Subsequently, in response to a query or update request from a Loom application that requires access to the DB, LIM parses the request and generates the appropriate data manipulation language (***DML***) statements for the DBMS; in the case of a query, it then processes the tuples returned to it by the DB into the form requested by the application. The details of the design and implementation appear elsewhere (Pastor & McKay 1994; Pastor, McKay & Finin 1992).

Processing within LIM is directed by a multi-layer KB architecture that is built in a mixed-initiative process. Figure 3 depicts the layers in this architecture. The Semantic Mapping KB **(*SMKB*)** is an isomorphic representation of the DB schema; it defines one Loom concept for each table and one Loom relation for each column. Application KBs (***AKB*s)** define view-concepts which are concepts or objects in the domain and refer to concepts and relations in the SMKB. Within the ARPI Information Agent, concepts such as *Seaport* are defined over underlying SMKB primitive data elements. View-concepts in the AKB do not necessarily map in any simple way to the tables in the DB, and can have arbitrary hierarchical structure. Connections to the DB are implemented via ***DB-mapping*** declarations, in which a concept-role pair in the AKB is mapped to a SMKB role. View-concepts are checked at definition time to assure that they specify an unambiguous database query and, if declared to be updatebale, are unambiguously so. For updates, LIM determines whether the resulting DB action should result in an insert or an update.

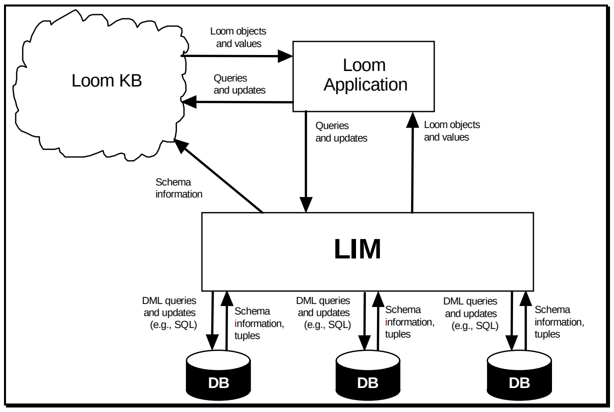
Loom KB
Loom objects and values
Queries and updates
Loom Application
Queries and updates
Loom objects and values
Schema information
LIM
DML queries and updates (e.g., SQL)
Schema information, tuples
DML queries and updates (e.g., SQL)
Schema information, tuples
DML queries and updates (e.g., SQL)
Schema information, tuples
DB
DB
DB

Figure 2. LIM Overview

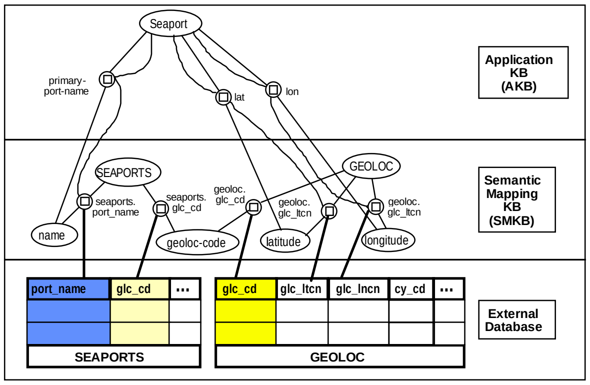

Figure 3. LIM Knowledge Base Architecture

LIM, given a query or update request involving a concept in the SMKB or AKB, first obtains schema mapping information from the SMKB, then translates the request into an equivalent DML statement, submits the statement to the DBMS and assembles the result; and finally (for a query), restructures the returned tuples as necessary, generating any KB structures required to satisfy the query. With regard to the last point, a fundamental principle of LIM is that KB structures are created only on demand: queries are satisfied without creation of KB objects whenever possible, to minimize overhead and bookkeeping. Control over object creation is entirely at the discretion of the application.

A LIM query consists of a list of output variables to be bound, and one or more statements that produce sets of bindings for these variables. It is easily determined from the positions of variables in the output list and the query expressions whether a particular output variable corresponds to a role value or a concept. For a variable corresponding to a role value, the value retrieved from the DB can be returned to the application, possibly with some conversion due to the differences between semantic types used in the KB and simple DB types. For a variable corresponding to a concept, however, the application will expect to have returned to it an instance of that concept; this requires that LIM be capable of creating Loom instances using values retrieved from the DB. LIM's object generation module extracts from the returned tuples all values requested specifically for the purpose of building Loom objects, creates the objects, and returns them to the application. In the Information Agent, the result, either a set of tuples or instance objects is then described in a KQML message using a content expression to desribe each tuple or object instance.

LIM uses a few different caching schemes, for two purposes. The first purpose is the conventional one of improving performance; the second is related to preserving referential integrity. When a user queries LIM for an instance of a view-concept, and then subsequently queries for an instance with the same key values, it is usually the case that one expects the *same* KB object to be returned in both cases. For this reason, LIM checks the Loom instance database (ABox) prior to creating instances. Given an object query, after submitting a query to the IDI and receiving return values, LIM queries the Loom ABox before creating a new instance. If the view-concept that is to be the type for the instance has keys defined, LIM uses these (in conjunction with Loom's indexing capabilities) to speed the search; otherwise, all values are used. This mechanism is also used to support incremental creation of object instances over several LIM queries. Other caching strategies avoid reissuing the same query.

Note that "ABox cache" checking is *not* an efficiency measure: on the contrary, it carries a performance penalty that can become significant on extremely large queries, e.g., many hundred to several thousand objects. For this reason, and because of situations such as dynamic DB contents where ABox cache checking is undesirable, it is controllable both globally and at the individual query level.

LIM Example

Let us presume that an application requires information about the location of various seaports. In the databases, information about seaports is stored in a table called `SEAPORTS`, and information about geographic locations in a table called `GEOLOC`. The various KB layers representing the mapping from application to DB are shown in Figure 3. The bottom panel shows a simplified tabular representation of the schema definitions for the two tables, `SEAPORTS` and `GEOLOC`. The middle panel shows the SMKB concepts representing the two tables. The SMKB definition is:

```
(defconcept Geoloc
  :is-primitive
  (:and db-concept
     (:the Geoloc.Glc_cd Geoloc_Code)
     (:the Geoloc.glc_lncn Longitude)
     (:the Geoloc.glc_ltcn Latitude)))
```

The top panel shows a simple application-level concept derived from information in both DB tables. The following is the Loom concept definition for the AKB concept seaport:

```
(defconcept seaport
  :is-primitive
  (:and View-Concept
     (:the primary-port-name string)
     (:the lat latitude)
     (:the lon longitude)))
```

This is mapped to the DB by making additional declarations, which are stored as assertions in the Loom KB. Queries can be posed referencing either the SMKB or the AKB. For example, the query:

```
(db-retrieve (?name)
  (:and
     (Seaports ?port)
     (Geoloc ?geoloc)
     (Seaports.Glc_cd ?port ?geocode)
     (Geoloc.Port_Code ?geoloc
?geocode)
     (Seaports.port_name ?port ?name)
     (Geoloc.Country_State_Code
?geoloc "DP")
     (Seaports.Clearance_Rail_Flag
?port "Y")))
```

("What are the names of seaports in Dogpatch that have railroad capabilities at the port?") can be posed using the SMKB. The SQL generated by LIM for this query is:

```
SELECT DISTINCT RV1.name
FROM SEAPORTS RV1, GEOLOC RV2
WHERE RV2.glc_cd = RV1.glc_cd
  AND RV2.country_state_code = 'DP'
  AND RV1.clearance_rail_flag = 'Y'
```

The values returned are a set of tuples:

```
("Cair Paravel" "Minas Tirith"
"Coheeries Town" "Lake Woebegon" "Oz")
```

The query:

```
(db-retrieve ?port
 (:and
    (seaport ?port)
    (primary-port-name ?port "Oz")))
```

("Return a seaport object for the port whose name is 'Oz'") can be posed using the AKB. The SQL generated for this query is:

```
SELECT DISTINCT RV1.name,
              RV2.latitude,
              RV2.longitude
FROM SEAPORTS RV1, GEOLOC RV2
WHERE RV2.glc_cd = RV1.glc_cd
  AND RV1.name = 'Oz'
```

The value returned by this query is an object whose Loom definition is:

```
(TELL
 (:ABOUT SEAPORT59253
    SEAPORT
    (LON 98.6)
    (LAT 3.14159)
    (PRIMARY-PORT-NAME "Oz")))
```

The Information Agent uses a slightly different form of the above s-expressions for sets of tuples and instances to return answers to other agents. The particulars are outside the scope of this paper.

Information Agent Performance

We have defined metrics for performance evaluation and have been using them continuously throughout the development of the Information Agent for both KQML and LIM. The performance model for KQML is described elsewhere. The LIM Information Agent metrics include components of total execution time:

- *Augmentation*: CPU time required to add concept-derived restrictions to the query
- *Translation*: CPU time required to translate LIM query into internal canonical form
- *Query Generation*: CPU time required to translate internal canonical form into DML
- *Connection*: Real time required to establish connection with DBMS server
- *Execution*: Real time required to execute the query on a DBMS server
- *Collection*: CPU time required to accumulate results of the query
- *Object Generation*: CPU time required to post-process results including creation of Loom instances if appropriate
- *ABOX Cache*: CPU time required to search Loom instance database (Abox) to prevent creation of duplicate instance (included in Object Generation time)
- *Total Execution Time*: Sum of all the above excluding the ABOX Cache time

Using benchmarks derived from queries collected during early uses of the LIM Information Agent under ARPI technology integration experiments, we have developed a performance profile. The queries vary from small functional tests to the retrieval of large view-concepts for force modules for combat services and combat services support; each force module is retrieved independently. The benchmark consists of executing the LIM query 25 times with all caching turned off, i.e., queries are sent to Oracle each time. Figure 4 below compares performance from initial baseline performance in November 1992, LIM 1.1 performance in May 1993, LIM 1.2 performance in May 1994, and LIM 1.4 performance in May 1995.

It should be noted that these queries retrieve and create a significant number of object instances with relatively large numbers of slot value sets; performance is now well below one second for total execution time. One test results in over 700 force module instances created and about 60,000 attribute value sets within those instances. The total execution time for this query set as of May 1994 was on the order of ten minutes; current execution time (LIM 1.4) is approximately 1 minute 25 seconds.

We have improved LIM performance dramatically over the course of the Planning Initiative. The most notable improvements are due to the following factors:

- We now use of faster Loom primitives where available, or adopted them they became available, which has dramatically improved basic execution speed.
- In cases where repeated use of the same Loom inferencing chain might otherwise result, we cache information retrieved from Loom knowledge base to "memoize" knowledge base access
- We use improved fundamental data structures within the LIM database interface
- We improved algorithms; for example, it is now possible to specify that results be returned from the Oracle interface in batches, rather than tuple-at-a-time.
- We tuned fundamental data structures extensively for speed and space.

All data has been collected on SUN SPARC 2 CPU with 96MB memory. Since a component of LIM processing is due to actual execution of queries on a remote Oracle database, the times measured below are dependent on the actual system used to support Oracle as well.

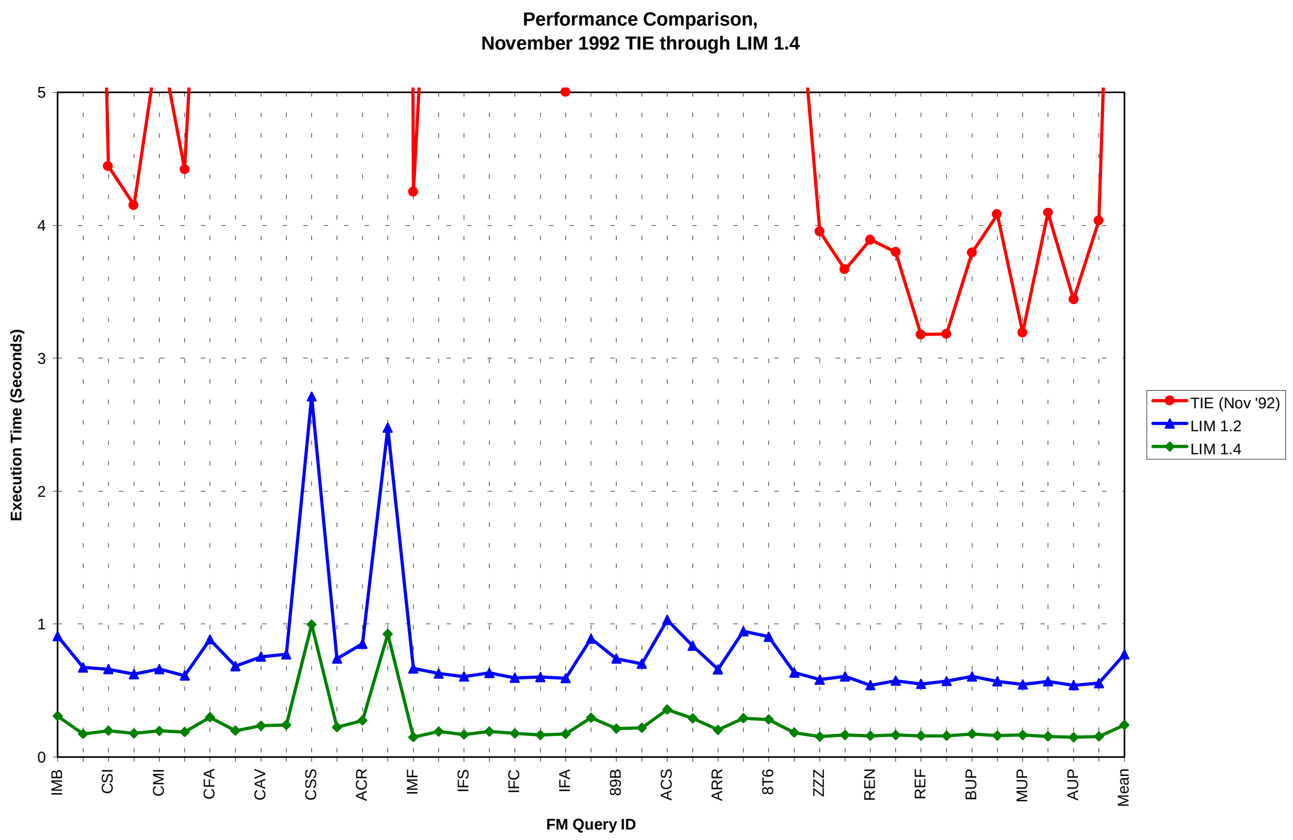

Figure 4 LIM Information Agent performance summary.

We have begun to compare critical portions of the LIM execution profile, specifically, object creation and slot filling algorithms, with those available in a commercial product expert system shell (ES). Initial results are preliminary, but, illustrate some of the performance issues for Information Agents. LIM, implemented in Common Lisp and Loom, outperforms the commercial ES, one written in C. We have measured the performance of LIM with that of the expert system shell (ES) on two query sets, showing both DB execution and Object Creation. The database and queries were selected from a set developed in another project. We have observed about a 10:1 ratio for object creations per second of DB execution time in favor of LIM. In addition, at least a 10:1 ratio for object creations per second of object creation time, again in favor of LIM. The most accurate metric is based upon slot-value sets, since this accurately reflects the total amount of data being transmitted and processed; in our initial measurements this is significantly over 10:1.

Conclusion

We have described an Information Agent architecture in which two key components are an agent communication language and a collection of information agents. Specifically, we have described KQML, the communication portion of the agent communication language. In addition, we have described the LIM Information Agent which interfaces the Loom knowledge representation and reasoning system with relational databases. We have described some of the performance measures we have developed for the LIM Information Agent and reviewed some of our current performance results. One set of preliminary measurements indicates that the performance of object creation and manipulation components for information agents is a key measure, and, that LIM outperforms at lest one widely available expert system shell. We intend to follow up on this result and investigate this measurement approach further.

The LIM Information Agent relies on a view-concept model which uses a knowledge representation language, Loom, to define the semantic schema of a database. This definition has two levels, each of which is of utility to a knowledge-based application. The semantic mapping layer defines the relevant concepts supported by the database domain; in our current knowledge bases, the semantic mapping layer adds semantic types to the automatically-generated schema model. We envision additional information in the semantic mapping layer, including composites of database objects which form larger conceptual structures. The view-concept model includes an application-specific layer that defines the mapping between an application domain's conceptual structures and the semantic definition of database concepts. We believe that the structured approach embodied in the view-concept model significantly elucidates the knowledge-base-to-database interface problem.

The system described above has operated in single module form where each of SIMS and CoBASE have LIM loaded into the same Common Lisp program, independently using LIM as a server remotely over the network (local or internet) and together in an architecture where SIMS acts both as a server to CoBASE and as a client to multiple LIM-based knowledge agents available on the network in a mixture of local and internet configurations. The basic issue addressed in all of the above work is the actual running of demonstrations in a reliable and repeatable manner. This goal forces one to pay attention to details of normal operations including performance and interpretation. Further, without the attempt to integrate, some of the issues described above would have not been identified as well as other integration issues.

Acknowledgments

The authors wish to acknowledge the contributions of the LIM development team, specifically Rebecca Davis, Robin McEntire, Rich Fritzson, Tim Finin, and Barry Silk (US Government). Earlier contributors to the first version of the Intelligent Database Interface include Tony O'Hare, currently at IBM, Research Triangle, and Larry Travis of the University of Wisconsin. Also, we acknowledge the useful comments and suggestions of users of LIM specifically Yigal Arens and Craig Knoblock of USC ISI, Wes Chu, Berthier Ribiero and Galdys Chow of UCLA, Scott Fouse, Nancy Lehrer, Mark Hoffman and Louis Rumanes of ISX, and, Glenn Abrett and Mark Burnstein of BBN. Finally, we have appreciated participating in the ARPA Rome Lab Planning Initiative and acknowledge the stimulating interactions with the participants in the program, Steve Cross and Tom Garvey the APPA Program Managers as well as the staff and program management at Rome Laboratory specifically Ray Liuzzi, Lou Hoebel, Don Roberts and Nort Fowler.

References

Yigal Arens 1992. Planning and Reformulating Queries for Semantically-Modeled Multidatabase Systems, In Proceedings of the First International Conference on Information and Knowledge Management.

Wesley W. Chu and Q. Chen 1994. A structured approach for cooperative query answering. *IEEE Transactions on Knowledge and Data Engineering*, 6(5):738--749.

Tim Finin, Richard Fritzson Don McKay and Robin McEntire 1994. KQML as an Agent Communication Language, In Proceedings of the Third International Conference on Information and Knowledge Management, ACM Press.

Tim Finin, Yannis Labrou, and James Mayfield 1995. KQML as an agent communication language, In Jeff Bradshaw (Ed.), ``Software Agents", MIT Press, Forthcoming.

Robert MacGregor and Raymond Bates 1987 The Loom Knowledge Representation Language, Proceedings of the Knowledge-Based Systems Workshop, St. Louis, Missouri.

James Mayfield, Yannis Labrou, and Tim Finin 1996. Evaluation of KQML as an Agent Communication Language. In Intelligent Agents Volume II -- Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages. M Wooldridge, J. P. Muller and M. Tambe (eds). Lecture Notes in Artificial Intelligence, Springer-Verlag.

R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. Swartout 1991. Enabling technology for knowledge sharing. AI Magazine, 12(3):36 -- 56.

Jon Pastor and Don McKay 1994. View Concepts - Persistent Storage for Planning and Scheduling, Proceedings of the ARPA/Rome Lab 1994 Knowledge-Based Planning and Scheduling Initiative Workshop, Tucson, AZ.

Jon Pastor, Don McKay and Tim Finin 1992. View-Concepts: Knowledge Based Access to Databases. In Proceedings of the First Conference on Information and Knowledge Management.

R. Patil, R. Fikes, P. Patel-Schneider, D. McKay, T. Finin, T. Gruber, and R. Neches 1992. The DARPA Knowledge Sharing Effort: Progress Report. In B. Nebel, C. Rich, and W. Swartout, editors, Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference.

Gio Wiederhold, 1986 Views, Objects, and Databases, *IEEE Computer*, 19(12):37–44.

Gio Wiederhold, 1992 Mediators in the Architecture of Future Information Systems, *IEEE Computer*, 25(3):38-49.