RL-TR-97-112
Final Technical Report
October 1997

PERSISTENT STORAGE TECHNOLOGY FOR PLANNING AND SCHEDULING

Lockheed Martin C2 Integration Systems

Sponsored by
Advanced Research Projects Agency
ARPA Order No. A007

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

19980223 112

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

DTIC QUALITY INSPECTED 3

Rome Laboratory
Air Force Materiel Command
Rome, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-97-112 has been reviewed and is approved for publication.

APPROVED: Raymond A. Liuzzi

RAYMOND A. LIUZZI
Project Engineer

FOR THE DIRECTOR: John A. Graniero

JOHN A. GRANIERO, Chief Scientist
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL/C3CA, 525 Brooks Rd, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

PERSISTENT STORAGE TECHNOLOGY FOR PLANNING AND SCHEDULING

Contractor: Lockheed Martin C2 Integration Systems
Contract Number: F30602-93-C-0028
Effective Date of Contract: 26 April 1993
Contract Expiration Date: 30 June 1996
Program Code Number: 0001
Short Title of Work: Persistent Storage Technology for Planning and Scheduling
Period of Work Covered: Apr 93 - Jun 96

Principal Investigator: Donald P. McKay
Phone: (610) 648-2256
RL Project Engineer: Raymond A. Liuzzi
Phone: (315) 330-3528

Approved for public release; distribution unlimited.

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by Raymond A. Liuzzi, RL/C3CA, 525 Brooks Rd, Rome, NY.

DTIC QUALITY INSPECTED 3

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Oct 97	Final Apr 93 - Jun 96

4. TITLE AND SUBTITLE

PERSISTENT STORAGE TECHNOLOGY FOR PLANNING AND SCHEDULING

5. FUNDING NUMBERS

C - F30602-93-C-0028
PE - 62301E
PR - A007
TA - 00
WU- 01

6. AUTHOR(S)

Jon A. Pastor, Donald P. McKay, and Robin McEntire

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Lockheed Martin C2 Integration Systems
Advanced Software Research/Advanced Information Systems
70 East Swedesford Road
Paoli, PA 19301

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, VA 22203-1714

Rome Laboratory/C3CA
525 Brooks Rd
Rome, NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

RL-TR-97-112

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: Raymond A. Liuzzi, C3CA, 315-330-1450

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release, distribution unlimited.

12b. DISTRIBUTION CODE

N/A

13. ABSTRACT *(Maximum 200 words)*

Knowledged-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 DARPA/Rome Lab Planning Initiative (ARPI), an intelligent information services architecture has been demonstrated which integrated cooperative user interaction and integrated information location via domain/user-oriented object representations. This effort involving participants and software components developed by Lockheed Martin, USC ISI and UCA 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, a DARPA-sponsored emerging language and protocol for information exchange.

14. SUBJECT TERMS

Computers, Artificial Intelligence, Software, Planning/Scheduling, Data/Knowledge Base

15. NUMBER OF PAGES

112

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Standard Form 298 (Rev. 2-89) (EG)
Prescribed by ANSI Std. 239.18
Designed using Perform Pro, WHS/DIOR, Oct 94

Table of Contents

1. SUMMARY 1

1.1 INTELLIGENT INFORMATION SERVICES FOR COLLABORATIVE PLANNING AND SCHEDULING 1
1.2 OBJECTIVE 1
1.3 APPROACH 2
1.4 PROGRESS 2
1.5 RECENT ACCOMPLISHMENTS 3
1.6 TECHNOLOGY TRANSITION 4
1.7 ACCOMPLISHMENTS 5
1.8 LESSONS LEARNED 7
1.9 DOCUMENTS PRODUCED 8
1.9.1 Government Documents 8
1.9.2 Technical Papers 8
1.10 MAJOR SOFTWARE SYSTEMS 9
1.11 SOFTWARE SITES 12

2. LOOM INTERFACE MODULE (LIM) 14

2.1 INTRODUCTION 14
2.1.1 Architecture 15
2.1.2 Semantic Mapping KB (SMKB) 15
2.1.3 Application KB (AKB) 15
2.2 OPERATION 16
2.2.1 Schema Generation 16
2.2.2 Schema Augmentation 16
2.2.3 Query Translation 17
2.2.4 Update 17
2.2.5 Object Generation 18
2.3 TECHNICAL DETAILS 18
2.3.1 Caching 18
2.3.2 Data Inference 19
2.3.3 Update Issues 20
2.3.4 Meta-View-Concepts 20
2.3.5 LIM Performance 27
2.3.6 Example 32
2.3.7 Conclusion 34

3. KNOWLEDGE QUERY AND MANIPULATION LANGUAGE (KQML) 35

3.1 INTRODUCTION 35
3.2 KQML 35
3.3 KQML PROTOCOLS 36
3.4 THE KQML LANGUAGE 37
3.5 KQML PERFORMANCE 38
3.6 LESSONS LEARNED 43

4. KNOWLEDGE SERVER ARCHITECTURE AND EXAMPLES 46

4.1 KQML AS A KNOWLEDGE SERVER LANGUAGE 48
4.1.1 What is an Agent Communication Language? 48
4.1.2 Desiderata 49
4.1.3 How KQML Stacks Up 50
4.1.4 Summary 52
4.2 AGENT NAME SERVICES AND THE ROLE OF THE AGENT NAME SERVER 53
4.2.1 Distributed agent-name resolution 54

4.2.2 *What do agent names look like?*....55
4.2.3 *Name resolution and Proxy Agents*....56
4.3 PLANNING INITIATIVE INFORMATION AGENT....56
4.3.1 *Technical Approach*....57
4.3.2 *LOOM Interface Module*....58
4.3.3 *Cooperative Information Access Research at UCLA*....60
4.3.4 *The SIMS Project at USC/ISI*....61
4.3.5 *Information Agent Lessons Learned*....64
5. KAKEE TIE....65
6. REFERENCES AND BIBLIOGRAPHY....69
6.1 LIM References....69
6.2 References for KQML....69
6.3 Information Agent References....71
7. PLANNING INITIATIVE REVIEW....73
7.1 Missed Opportunities....73
7.2 Structural Model....74

Figure 1.....15
Figure 2.....16
Figure 3.....18
Figure 4.....22
Figure 5.....27
Figure 6.....30
Figure 7.....30
Figure 8.....31
Figure 9.....31
Figure 10.....37
Figure 11.....37
Figure 12.....38
Figure 13.....38
Figure 14.....40
Figure 15.....41
Figure 16.....42
Figure 17.....42
Figure 18.....43
Figure 19.....43
Figure 20.....44
Figure 21.....45
Figure 22.....45
Figure 23.....46
Figure 24.....55
Figure 25.....59
Figure 26.....62

1. SUMMARY

1.1 Intelligent Information Services for Collaborative Planning and Scheduling

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), , in collaboration with USC ISI and UCLA, demonstrated an intelligent information services architecture has been demonstrated which integrated cooperative user interaction and integrated information location via domain/user-oriented object representations. This effort involving participants and software components developed by Lockheed Martin, 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.

TECHNICAL FOCUS

APPROACH:

- Develop common interfaces and representations for access to relational and object-oriented databases from LOOM and to intelligent agents using KRSL and shared domain definitions.

ISSUES:

- Persistent integration of data and knowledge stored in distributed, heterogeneous systems including databases and intelligent agents

TRADE-OFFS:

- Common representations supporting multi-system interactions vs isolated and application-specific representations
- Multiple interface layers support information hiding and promote modularity vs single-purpose tailored interfaces.

PROBLEMS:

- Multi-layer systems add overhead - performance being addressed
- Persistent "plans" requires shared PI definitions - SDO being developed
- Many possible multi-agent, multi-knowledge server systems - ARPI multi-agent systems being defined in PI CPE, TIEs and IFDs

1.2 Objective

Lockheed Martin Tactical Defense Systems (LMTDS) is developing 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. These components, when integrated with other intelligent

information system components being developed at USC ISI (SIMS) and at UCLA (CoBASE), will provide intelligent access to distributed information sources in a fault tolerant and cooperative manner supporting military planners.

1.3 Approach

Lockheed Martin is developing and demonstrating technology components providing intelligent access to distributed information sources including databases and knowledge-based systems. Demonstrations are based upon providing intelligent information services identified by military planners as desired capabilities lacking in current systems. The integration of knowledge bases and databases is accomplished by extending the current prototype implementation of the LOOM Interface Module (LIM). LIM allows LOOM 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, Lockheed Martin is designing and prototyping a new high-level protocol for conveying knowledge between systems. This protocol, KQML (Knowledge Query and Manipulation Language), is being developed in conjunction with the ARPA Knowledge Sharing Initiative. The Lockheed Martin Planning Initiative effort supports experimental use of KQML for intelligent information access under technology integration experiments and is not a principal deliverable of this contract. A separate effort under the ARPA I3 program supports development of the language and software system.

FOCUS

GOALS:

Persistent storage of SDO definitions, e.g., Common Plan Representation (CPR), COAs, high-level and detailed employment and deployment plans, schedules, etc.

Promote runtime sharing of data and knowledge via shared knowledge and data bases using common language for communication and shared domain representations.

BASELINE:

Some "standard" SDO definitions mapped to data base representations for query and simple update, e.g., geographic locations, assets, etc.

No persistence for emerging common plan representation

Experimental sharing of knowledge - individual systems have private knowledge and data (redundant, inconsistent, ad hoc)

EXPECTED RESULTS:

Data base representations for common plan representation

LIM: Standard mapping of PI domain ontology to data bases with persistence

Knowledge sharing in multi-agent systems

1.4 Progress

Lockheed Martin achieved significant progress in providing convenient, *efficient* access to remote databases for Loom applications. We focused heavily on performance enhancement: as a result of our efforts, current LIM/IDI performance on our benchmark query suite (based on actual queries from the ARPI FMERG TIE in November 1992) is 40 times better on average, and 65 times better in the worst case, than the first trials with these data. To restate these results for the sake of clarity, the current mean total execution time is approximately 2.5% of the original total execution time, and the worst-case total execution time is approximately 1.5% of the original total execution time: all queries execute, from query

issuance to return of resulting Loom instances, in under one second, and the mean is approximately 240 milliseconds. On our secondary benchmark suite (derived from the ARPI KAKEE/FORMAT TIE in February 1994), performance is over 4 times better on average, and almost 10 times better in the worst case, than in the first trials with these data: average total execution time is now on the order of 2.0 seconds, with only the largest five queries (which return hundreds of objects, and require thousands of slot-value sets) in the test suite taking more than 2.0 seconds, and most (65%) taking well under 1.0 seconds.

1.5 Recent Accomplishments

Our principal focus in this stage of LIM development was on making LIM a practical tool for use by other systems. This includes dramatic improvements in performance, added features that extend its capabilities, and a significant contribution to the integration of LIM into other systems.

STATUS

Significant Accomplishments:

Support for DDL added

Partial instantiation finished

Performance improved dramatically

near-O(2) improvement relative to baseline

typical queries processed in 2 seconds or less

Future Work:

Additional domain and persistent representations

Expand coverage of domain (CPR, intermediate results)

Expand functionality and improve performance

Distributed multi-knowledge server systems

Eliminate single-agent knowledge server bottleneck

Develop additional common problem sets for intelligent information access experiments

- **Performance**:
  - We have compared critical portions of the LIM execution profile, specifically, object creation and slot filling algorithms, with those availble in a commercial product expert system shell. LIM, implemented in Common Lisp and Loom, outperforms the commercial product, one written in C, by almost a factor of 100x.
  - We further improved LIM/IDI performance; at this point, we have improved average-case performance by a factor of almost 40, and worst-case performance by a factor of over 60, since the original benchmark on the TIE query set.
  - We revised and enhances our performance and benchmarking tools to permit capturing a wider range of performance measures and controlling performance monitoring with fine resolution.
- **Features:**
  - We added DDL capability to LIM and the IDI: it is now possible to create and destroy tables, and to add and drop indices on tables, via simple statements in the LIM query/update language
  - We completed the implementation of partial object instantiation, a capability that permits the user to specify which slots of newly-retrieved objects should be filled.
  - We ported both LIM and the IDI to Allegro Common Lisp, a port which will make both systems more accessible to end-users, since ACL has become a leader distributor of Lisp environments.
  - We ported the Lisp API of KQML to Allegro Common Lisp, a port which will make KQML more accessible to end-users, since ACL has become a leader distributor of Lisp environments.
- **Integration, Collaboration, and Support:**

- We assisted Rome Lab staff in installing and integrating LIM, CoBase, and SIMS using the same databases, knowledge bases, and software releases.
- We began the process of restructuring the knowledge-bases and Lisp packages used by LIM to facilitate and simplify future integration with other components.
- We continued to support demonstrations by collaborators in the KB/DB group, including several live remote SIMS demonstrations by USC ISI.
- We exported the Lockheed Martin SDO for use with SRI's GKB knowledge editor, and performed two rounds of evaluation for them using the SDO KB; we also provided this KB to staff at Rome Lab for their use.
- We provided the Allegro Common Lisp port of the IDI to Dr. Peter Karp's group at SRI for use in their access to biological databases.
- We began experiments with the Stanford World Wide Web interface to Ontolingua, with an eye toward possible use in development of ontologies, and possible future LIM attachment.
- We provided the Lockheed Martin implementation of KQML to the SIPE group at SRI for use in a collaborative planning experiment and evaluation. KQML was selected by the SRI group as the language of choice for implementing collaborative agent-based planning systems.

♦ **Miscellaneous:**

- We prepared a synopsis of the work to date on LIM and its use in TIEs with USC ISI and UCLA for inclusion in the IEEE Expert special issue on the Planning Initiative.
- The Valley Forge Engineering Center has received ISO 9000 certification by an official ISO registration audit team. In preparation for this visit, we prepared a number of web-based project directories of information including reports, presentation, documentation, etc..
- We participated in and contributed to several proposals. These include ARPA proposals for integrating software engineering and ontology development and a proposal to the Logistics R+D program. Also we contributed to a proposal for an Enterprise Integration Toolkit (EIT) for healthcare information networks selected by NIST ATP. Both proposals were awarded, largely on the strength of our architectures and methodologies.

1.6 Technology Transition

The IDI, LIM and KQML as well as the underlying military transportation databases have been used extensively within the ARPI Common Prototyping Environment (CPE) and Technology Integration Experiments. All are components in the current CPE and are significant candidates for the future ARPI shared tools. USC ISI's SIMS multi-source information access planning system and UCLA's CoBASE cooperative database system use LIM, the models, databases and sample queries developed by Lockheed Martin as their underlying representation and database interface. Mitre's active database prototype system, the Mitre/ISX force module definition prototype, the UCI information integration project and SRI very large knowledge-base efforts utilize the IDI to support database access.

Outside of the ARPI community, our technology contributed to the awarding of contracts to Unisys Health Information Management for an Enterprise Integration Toolkit (EIT) under NIST ATP, and to Lockheed Martin by the State of Wisconsin for a Medicaid information system (KUMA). We contributed significantly to both proposal efforts, and expected to participate in the realization of the designs on which we collaborated.

Key technologies exported from this project include intelligent interfaces to databases and interagent communication.

Systems -

- LOOM Interface Module (LIM) - LIM supports an interface to relational databases for the LOOM knowledge representation and reasoning system. LIM runs in Lucid Common Lisp using Oracle on SPARC 2 UNIX platforms with a minimum of 32MB of physical memory. POC: Jon Pastor, Lockheed Martin Tactical Defense Systems, (610) 648-2769, pastor@vfl.paramax.com.

- Intelligent Database Interface (IDI) - IDI defines a generic interface to databases from Common Lisp currently supporting Oracle and Ingres. The IDI runs in Lucid Common Lisp on SPARC 2 UNIX platforms with a minimum of 32MB of physical memory. POC: Robin McEntire, Lockheed Martin Tactical Defense Systems, (610) 648-2191, robin@vfl.paramax.com.

- Knowledge Query and Manipulation Language (KQML) - KQML defines a common language and protocol for intelligent agent communication and its development is supported via the ARPA Knowledge Sharing Initiative. KQML implementations exist for Common Lisp (Lucid) and C running on SPARC 2 UNIX platforms with a minimum of 16MB of physical memory. POC: Don McKay, Lockheed Martin Tactical Defense Systems, (610) 648-2256, mckay@vfl.paramax.com.

1.7 Accomplishments

In this report, we present the basic accomplishments achieved under this contract which include:

- Software systems and associated documentation for knowledge base access to databases (the LOOM Interface Module and the Intelligent Database Interface) and for communication among multi-agent systems (the Knowledge Query and Manipulation language).
- Performance metrics and instrumentation for knowledge base access to databases which highlight performance improvements.
- Software supporting the KB/DB Technical Working Group and the development of joint demonstrations.
- Military transportation logistics knowledge bases and databases for use within the Planning Initiative.
- Technology Integration Experiments and demonstrations at Rome Laboratory and at the Planning Initiative Workshops.

The Intelligent Database Interface (IDI) acts as an intermediary between a knowledge-base and one or more external databases. In addition to allowing more than one connection to an external database, the IDI allows connections to different DBMSs (currently they must be relational DBMSs) and allows different methods of making those connections. It can be used as a software library to interface Common Lisp applications to databases, either external or internal, as a standalone software system via its logic-oriented query language or as an embedded system, for example as part of the LIM system supporting a knowledge server. The IDI is in principle use within the Planning Initiative in this embedded form, however, a few sites have been interested in the IDI as a standalone or embedded system including MITRE, the University of Massachusetts, USC ISI, Georgia Tech, MIT Sloan School and SRI.

The LOOM Interface Module (LIM) is designed to provide convenient access to DBs for knowledge-based systems that use the Loom knowledge-representation system, using a multi-layer architecture based on the IDI. The present release of LIM supports retrieval of both values and objects, including hierarchically-structured objects, as well as update for objects. The View-Object model is an extended entity-relationship model that maps relational tables into hierarchical objects; LIM extends this further by mapping to the semantically-rich structures of a frame-based knowledge-representation system. All processing in LIM is driven by a multi-layer Loom model of DBs, applications, and the mappings between them. LIM generates a Loom model of the schema for each DB. A Knowledge Base Administrator (KBA), whose role is analogous to that of a DBA for databases, then defines Loom concepts in the application domain, and describes the mappings from roles of these to roles of concepts in the schema model. When a query is posed against an application concept that has been mapped to the DB, LIM translates this into a query against the DB, submits it, retrieves the result, and does any post-processing necessary before returning the answer to the application that issued the query.

The advantage of the LIM architecture is that application writers do not need to know anything about:

- tables, columns, names, types, or other attributes of DB structures;
- DBMS open, connection, and close protocols; or
- DBMS query languages.

Furthermore, Loom objects created by LIM are indistinguishable from other knowledge-base objects, and applications need only be aware that a given object is derived from the DB if it must be updated.

These basic ideas have been tested and proven effective in two classes of applications:

- a knowledge server which maps knowledge structures from relevant portions of the Planning Initiative Shared Domain Ontology (SDO) onto standard military transportation databases
- a Planning Initiative Information Agent system which integrates the capabilities of LIM with those of CoBASE (UCLA) and SIMS (USC ISI).

In the case of the knowledge server, access to military transportation knowledge bases and databases was made transparent to applications such as the FMERG force module expansion system. For example, FMERG requires basic information about the combat support and combat service support units associated with a particular combat unit. In the Infrastructure Prototype Technology Integration Experiment done as a joint effort among Lockheed Martin, BBN, ISX, SRI and GE CRD, FMERG retrieved this force structure and expansion information from a knowledge server using SDO definitions. The knowledge server, via LIM, mapped queries against these SDO definitions to an Oracle database when the information resided in databases. Further, FMERG was insulated from the actual storage location as was demonstrated when database information was cached in the knowledge server's instance-level knowledge base.

For the Information Agent, several different configurations have been developed and demonstrated:

- CoBASE is a cooperative database system which can relax queries based on a semantic model of the domain represented in LOOM, e.g., relaxing specific geographic coordinates to regions or areas. In August of 1992, a system integrating CoBASE and LIM was demonstrated in which LIM was used as an embedded query processor.
- SIMS is a multi-source information planning system used, in its current application to the Planning Initiative, as an information access planner to multiple databases. In the summer of 1992, SIMS was demonstrated using LIM as its query processor for each individual database.
- At the second Planning Initiative Workshop, these systems were demonstrated along with a third system which integrated CoBASE, SIMS and LIM to demonstrate an integrated Information Agent

capable of providing cooperative responses to user queries (CoBASE) supported by multiple databases (SIMS) which was based upon the Planning Initiative SDO mapped to Oracle databases (LIM). This system was developed and integrated in a relatively short amount of time since all systems were already using the LOOM knowledge representation system and its query language as well as SDO and database components already having been developed and disseminated for use by this group under Lockheed Martin effort.

- At the third Planning Initiative Workshop in February, 1994, the Information Agent architecture had evolved to an agent architecture where each component, LIM, CoBASE and the SIMS was an independent software process. Each was accessible via KNET using the Lockheed Martin KQML implementation. This architecture operates over a local area network or over the internet in a location transparent manner. The architecture can be manipulated to local in one configuration and then quickly changed to an internet demonstration. Several demonstrations of this architecture using SIMS and CoBASE were given during January 1994 through May 1994.

Rome Lab staff recently installed the three components described above in their facility. At that time, each system used its own knowledge bases and associated declarations for standalone use, and we were consulted on the integration of these. We produced a consistent set of files that can be shared among all three applications, even when each is used independently. Finally, it was demonstrations of the CoBase/SIMS/LIM/IDI system that finally convinced Unisys staff of the validity and value of the architecture that we had proposed to them for EIT.

We continued to concentrate on performance for the combination of the LIM/IDI, based upon the test suite of queries we have developed, a set of queries generated by the FMERG system as a part of the Infrastructure Prototype TIE, and a set of queries generated by the KAKEE TIE in February and March 1994. For these evaluations, we compared performance from the November 1992 FMERG TIE and the February 1994 KAKEE TIE with performance of the LIM 1.3 and LIM 1.4 systems after performance improvements were made to each of the systems. The results are described in detail in below. We believe that including queries generated by hand as well as those from the actual execution of CPE (FMERG) and potential CPE components (FORMAT) provides a representative test set. The basic summary of the performance results are:

- for FMERG queries
  - mean total execution time has been reduced by a factor of approximately 40x, with a range of approximately 20x to 72x
  - worst-case performance has been improved by a factor of approximately 65x
  - all queries execute in less than 1.0 second, with a mean total execution time of approximately 240 millliseconds
- for KAKEE queries
  - mean total execution time has been reduced by a factor of approximately 4.25x, with a range of approximately 2.8x to 9.8x [recall that the original KAKEE results are just over one year old, and thus substantially better than they would have been in November 1992]
  - worst-case performance has been improved by a factor of approximately 4.6x
  - all but the largest 5 queries execute in less than 2.0 seconds, with a mean total execution time of approximately 2.0 seconds; the five exceptions all have over 100 (104-711) component objects, and require over 8,000 (8,139-56,092) slot-value sets

1.8 Lessons Learned

The primary lessons learned are:

A particularly critical enabling issue is that of data, knowledge and use of the data and knowledge (e.g., queries). The Knowledge Base / Data Base Technical Working Group established at the Planning Initiative kick-off meeting in February of 1991 identified the lack of coherent data sets, in database form or any form, as well as queries to be a severe limitation to demonstrating the key ideas of the technical groups involved. To this end, Lockheed Martin has led the effort in the Planning Initiative in providing domain data in the form of usable databases, knowledge bases and fragments of the SDO from which retrieve information from the databases. The success, and importance, of this effort can be measured by the number of sites and systems using the data and databases in their day to day research and development efforts. The initial efforts yielded examples and small use scenarios that supported significant demonstration of the SIMS and CoBASE systems for over three years of continuous research and development within the Planning Initiative. Further, Lockheed Martin has used this data to establish performance test suites for its own internal development efforts as reported here.

While providing the underlying data, knowledge and examples is important for the research and development tier, the application of the resultant technology is dependent on application development issues such as basic applicability of the technology, scalability and performance. To this end, Lockheed Martin has participated in the Technology Integration Experiment (TIE) efforts to understand first hand some of these issues and feed them back to the research community as additional test and performance suites. As a first step, Lockheed Martin has supplied the same databases that are used in the Infrastructure Prototype TIE to the research groups at USC ISI and UCLA for use in their individual development efforts as well as other collaborative work. For Lockheed Martin efforts, the participation in TIEs has been fruitful to the extent application requirements become identified under the aegis of developing integrated prototypes. We believe that the performance issues uncovered during the TIE process provided an essential component to our development work which would otherwise been unavailable to us. The performance evaluation section of this report is a direct outcome of Lockheed Martin participation in the TIEs. The conclusion that we reached after analysis of the KAKEE results is that doing anything non-trivial 60,000 times is costly.

1.9 Documents Produced

1.9.1 Government Documents

The following documents describe the software developed under this contract:

1. Software Design Document for the LOOM Interface Module (LIM) and Intelligent Database Interface (IDI)
2. Software User's Manual and for the LOOM Interface Module (LIM) and Intelligent Database Interface (IDI)
3. Computer Systems Operations Manual for the LOOM Interface Module (LIM) and Intelligent Database Interface (IDI)

1.9.2 Technical Papers

The following technical papers were developed or contributed to under this contract:

Contributions to Planning Initiative IEEE Expert Issue.

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

Jon Pastor, Don McKay and Tim Finin, "View-Concepts: Knowledge-Based Access to Databases," *First International Conference on Information and Knowledge Management*, Baltimore, November 1992.

Tim Finin, Don McKay, Rich Fritzson, and Robin McEntire, "KQML: An Information and Knowledge Exchange Protocol" , in Kazuhiro Fuchi and Toshio Yokoi (Ed.), "Knowledge Building and Knowledge Sharing", Ohmsha and IOS Press, 1994.

Tim Finin, Don McKay, Rich Fritzson and Robin McEntire, KQML - A Language and Protocol for Knowledge and Information Exchange. *Proceedings of the 13th International Distributed Artificial Intelligence Workshop.*

Tim Finin, Rich Fritzson and Don McKay, A language and protocol to support intelligent agent interoperability. In *Proceedings of the CE and CALS Washington '92 Conference*, June, 1992.

Ramesh Patil, Richard Fikes, Peter Patel-Schneider, Don McKay, Tim Finin, Thomas Gruber, Robert Neches, The DARPA Knowledge Sharing Effort: Progress Report. In Proceedings KR '92, Boston, MA, November, 1992.

1.10 Major Software Systems

During the reporting period, we have developed two major releases of LIM. The release notes described the basic features of the software. The software is available to Planning Initiative participants under the ARPI internal technology sharing agreements.

LIM Release 1.1

- Loom 2.0 compatibility: LIM 1.1 has been tested under Loom 2.0 (final) through patch-level 12.
- Due to incompatible changes between Loom 2.0b2 and Loom 2.0(f), LIM 1.1 as distributed cannot be used with Loom 2.0b2; if it is impossible for you to switch to Loom 2.0, please contact us for advice.
- Logical pathnames: In conformance with Common Prototyping Environment (CPE) standards, LIM now uses Common Lisp logical pathnames in the specification of all filenames, as well as using the standard CPE defsystem. We have included the CPE files PE defsystem.lisp (in the sub-directory ./utils) and logical-pathnames.lisp (in the top-level IDI directory) in the IDI 2.4 release for your convenience ; if you require any assistance in using them, contact the support personnel for the ARPI CPE.
- Use of Loom Key- and Index-roles: Loom 2.0(f) introduced the notions of key- and index-roles, where the former specify sets of roles that are constrained to have unique values for all instances, and the latter specify sets of roles on which retrieval indices are maintained.

- Improved performance for object queries: object identifiers and changes to A-Box "cache" checking: LIM can now be told to generate identifiers for any Loom objects it creates, and to use these identifiers for retrievals in the Loom Abox wherever possible. This is possible only for instances of concepts for which LIM key-roles have been defined, and the identifier is the concatenation of the concept-name and the key-values for an instance; the order of the key-values is canonicalized in a manner that guarantees that the same identifier will always be generated for the same set of key-values. Use of identifiers is strongly encouraged, as it results in the best possible performance for LIM Abox checking; however, if identifiers are not used, instances of concepts with LIM key-roles defined will have a significant performance advantage over those without LIM key-roles defined, since LIM key-roles imply Loom indexing on those roles. Even in the case of concepts without key-roles, retrieval is significantly faster in LIM 1.1 than in previous releases due to the use of Loom retrieval functions publicized since the LIM 1.0 release. The net result of all of these changes is an improvement in performance that ranges from moderate to significant, depending on the number of objects returned by a query. For queries returning a large number of objects (say, 100 or more), the improvement is dramatic: relative to performance in November 1992, total execution time has decreased by an order of magnitude or more for such queries.

- Faster startup (IDI): Release 2.4 of the IDI incorporates changes that speed up reading of DB schema information substantially; schema-read times under IDI 2.4 should be on the order of 15-25% of schema-read times under IDI 2.3.

LIM 1.2

LIM 1.2 includes a number of significant additions and improvements relative to earlier releases:

- support for update of all View-Concepts, including those involving joins and those having DB-derived sub-parts
- support for Meta-View-Concepts, permitting class-level information from a DB to be retrieved and automatically asserted about View-Concepts representing those classes
- improved performance
- support for Loom 2.1
- support for Lucid CL 4.1.1

In addition to data about individual real-world objects like seaports and ships, the ARPI databases contain data about *classes* of real-world objects. Examples of this include data about the dozens of types of ships that are recognized, and about the characteristics of a dozen or so types of berths. The Shared Domain Ontology (SDO) currently represents this information explicitly in the knowledge base files. It is highly desirable that information about classes of objects be accessible from the context of the objects themselves. In LIM 1.2, we have implemented a meta-view-concept facility for LIM that provides the ability to define meta-level view-concepts, and to retrieve data for instances of these — i.e., View-Concepts — from the database (see Section 2.3.4). Complete details are described in the LIM 1.2 Software Design Document and the Software User's Manual.

LIM 1.3

The principal features of LIM 1.3 were:

- Performance: average performance has improved significantly in LIM 1.3 relative to previous releases, in terms of both execution time and memory consumption. The initial query for a given concept (not instance) typically takes approximately the same time as under LIM 1.2, but subsequent queries for the same concept are 15-30% faster.
- Memory consumption ("Dynamic Bytes Consed" and "Ephemeral Bytes Consed", as reported by the LCL TIME function) has been reduced by 25-50%, with savings proportional to the number and size of the objects created.
- A new release of IDI has been distributed concurrently with this LIM release. Some of the performance improvements in LIM are due to features of this release of IDI.

LIM 1.4

The principal features of LIM 1.4 were:

- further improved performance, particularly for large queries (e.g., the largest KAKEE force modules)
- compatibility features to support the integration of LIM, CoBASE and SIMS; these were made with the cooperation of UCLA and ISI staff, primarily for the benefit of Rome Lab's installation of our systems for demonstration and internal use.

LIM 2.0

The principal features of LIM 1.4 were:

- further improved performance, particularly for large queries (e.g., the largest KAKEE force modules)
- compatibility features to support the integration of LIM, CoBASE and SIMS; these were made with the cooperation of UCLA and ISI staff, primarily for the benefit of Rome Lab's installation of our systems for demonstration and internal use.

For a discussion of LIM performance in general, see Sections 2.3.5 and 3.

1.11 Software Sites

The software systems developed under this contract has been in use within the Planning Initiative for well over the past two years. These were early interim releases of the software that were made available to obtain early use as well as feedback to guide further development. The approach, although not always successful, has aided substantially in the rapid development of the software especially with regard to performance. These are the principle sites within the PI where projects have used the indicated software.

There is use of the software outside of the Planning Initiative within the Intelligent Integration of Information program at SRI, UCI, USC ISI, UCLA and Georgia Tech.

The following sites are using, plan to use or are investigating the use of the ***Intelligent Database Interface*** either standalone or as an embedded system:

- MITRE
- USC ISI
- UCLA
- SRI
- UCI
- Georgia Tech
- University of Massachusetts

The following sites are using, plan to use or are investigating the use of the ***LOOM Interface Module*** either standalone, as an embedded system or as a part of a knowledge server:

- ISX
- BBN
- USC ISI
- UCLA
- MITRE

The following sites within have used ***KQML*** successfully to integrate one or more systems or are investigating the further use of KQML:

- BBN/ISX
- UCLA
- USC ISI
- Mitre

2. LOOM INTERFACE MODULE (LIM)

2.1 Introduction

The integration of artificial intelligence (***AI***) and database management system (***DBMS***) technologies promises to play a significant role in shaping the future of computing. As noted in [7], AI/DB integration is crucial not only for next-generation computing, but also for the continued development of DBMS technology and, in many cases, for the effective application of AI technology. The motivations driving the integration of these two technologies include the need for

- access to large amounts of shared data for knowledge and information processing,
- efficient management of knowledge as well as data, and
- intelligent processing of data.

In addition, AI/DB integration at Lockheed Martin was motivated by the desire to preserve the substantial investment in most existing, or ***legacy***, databases. To that end, a key design criterion was that our integration technology support the use of existing DBMSs as independent system components. Finally, the development of large-scale, heterogeneous, distributed AI systems involving a number of discrete cooperating agents, as well as a general need to retain results of AI processes for the long-term, have led to a need for ***persistent storage*** of knowledge to permit ***knowledge sharing*** among those agents.

Some work has been done toward implementing a ***view-object*** model [15] on relational DBs for which a semantic schema exists. In particular, Barsalou and Wiederhold [3,4] describe a system based on the ***Structural Model*** [14], an extended entity-relationship model. In this system, the user selects a ***pivot*** relation that includes the intended key(s) for the object being defined. The various link-types in a ***structural schema*** are then traversed, and, using a relevancy metric, a tree of candidate relations—rooted at the pivot—is generated. The user then prunes this tree, leaving only the relations and attributes that s/he wishes to have in the defined object. Once the object is defined, it is linked into an object hierarchy. Barsalou and Wiederhold provide algorithms that assure that objects are retrievable, and, if desired, updatable. The ***View-Concept*** model described here is intended as a knowledge-based extension of the view-object model; it also draws upon work done at Lockheed Martin on an Ingres DB interface for the CYC KRS [12] and on the Cache-based Intelligent Database Interface (***CIDI***) [11]. We have implemented a prototype system for LOOM called the LOOM Interface Module (***LIM***).

LIM is being developed in the context of the ARPA/Rome Lab Planning Initiative (PI) a multi-site project whose goal is the development and introduction of a knowledge-based system to support military logistics planning for the United States Transportation Command, which is responsible for planning large-scale movements of troops and materiel. The current planning system is a 1970s-vintage system that uses large, heavily-encoded records; the plan for even a moderate-sized operation is enormous, with some containing hundreds of thousands of records, effectively precluding any purely memory-resident storage scheme. The system has recently been extended to use a relational database in place of flat files for some applications, and in some cases, object oriented databases. The designers of the relational DB schema were constrained by a need to adhere fairly closely to the original data format for the plan records, since the planners are quite familiar with the current structure of the data. Our project is thus faced with the task of interfacing a knowledge representation system to a collection of legacy databases that lack a coherent semantic schema. There is also a need to share data derived by one among a group of AI systems with other systems, either in real-time or at a later time.

Related work within the ARPI is being performed by groups at ISI [1,2] and UCLA [8]. ISI's ***SIMS*** (Services and Information Management for Decision Systems) is designed to map the queries of users, who are presumed to be ignorant of the structure and content of a collection of databases, into retrievals

against those databases. UCLA's ***COBASE*** (Cooperative Database) is a knowledge-based extension to standard relational query languages (e.g., SQL) that provides approximate operators supporting query relaxation and approximate answers.

2.1.1 Architecture

LIM acts as an intermediary between a Loom application and one or more DBs, using the services of the IDI to access the DBs. The inter-relationships among the various components of the overall system are illustrated in Figure 1. LIM uses the IDI to read 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 uses the IDI to generate 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 IDI into the form requested by the application.

Processing within LIM is directed by a multi-layer KB architecture that is built in a mixed-initiative process. Figure 2 depicts the layers in this architecture.

2.1.2 Semantic Mapping KB (SMKB)

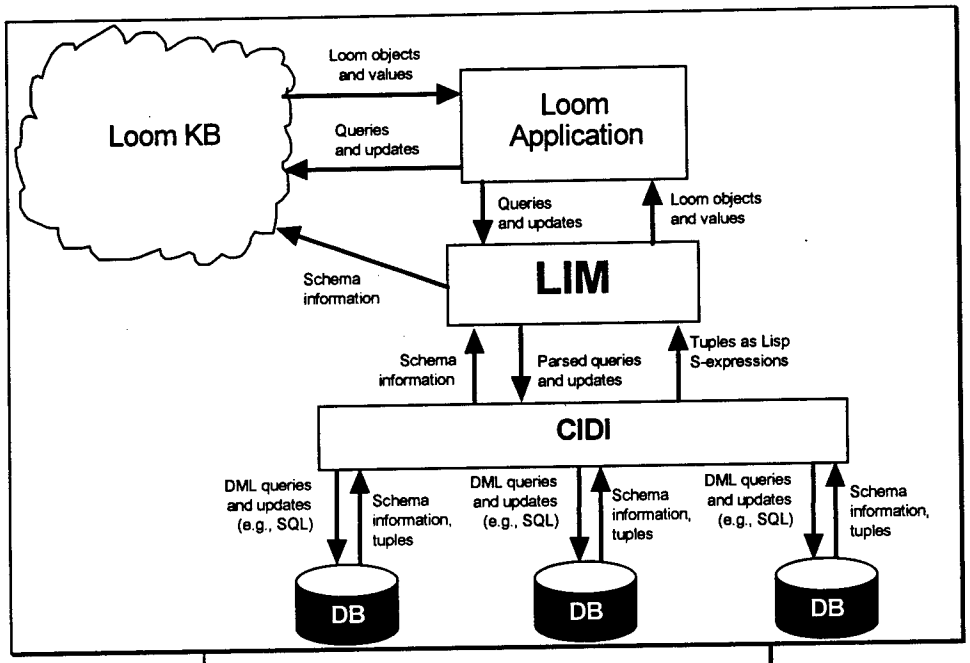


Figure 1. LIM Overview

The Semantic Mapping KB (***SMKB***) is an isomorphic representation of the DB schema. It is constructed by a Knowledge Base Administrator (***KBA***) — analogous to the database administrator for a DB — from an automatically-generated schema model. LIM creates this schema model by using the IDI to read the DB schema; based on this information, it defines one Loom concept for each table and one Loom relation for each column. The relation representing each column is defined as a role on the concept representing the appropriate table, value-restricted to a type that represents the DB type (domain) of the column; these DB types are typically simple types like `integer` and `string`. The SMKB is created primarily by defining *semantic* types, and then substituting these for the simple DB types that appear in the schema model. LIM stores information identifying the DB, table, and column with the concepts and relations in the SMKB.

2.1.3 Application KB (AKB)

Application KBs (***AKB***s) refer to concepts and relations in the SMKB. Unlike concepts in the SMKB, which are isomorphic to tables in the DB, and which presently have no hierarchical structure, 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.

2.2 Operation

LIM, given a query or update request involving a concept in the SMKB or AKB,

- obtains schema mapping information from the SMKB;
- translates the request into an equivalent DML statement with the aid of the IDI, which submits the statement to the DBMS and assembles the result; and
- 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.

The processing performed by LIM is illustrated in Figure 3.

2.2.1 Schema Generation

As described above, when a DB is opened via the IDI, schema information is cached by the IDI in local data structures. The schema generation module reads this information, and generates one Loom concept per table and one Loom relation per column. The relations corresponding to the columns of a table are then added as roles of the corresponding concept via value-restrictions.

2.2.2 Schema Augmentation

The automatically-generated schema model is a literal representation of the DB schema. One implication of this is that the semantics of relationships among tables are not explicit, since the schema does not identify the columns in the DB over which joins are semantically reasonable. In creating the SMKB, the KBA augments this literal schema representation by defining semantic types to explicate the semantics of joins. In particular, where two relations represent columns over which a join is semantically reasonable, the value restrictions on these relations in their respective concepts are changed to the same KB type. For example, two DB columns whose DB type is `integer`, but which both represent a particular kind of identification number, would have their value restrictions specialized to a concept representing that

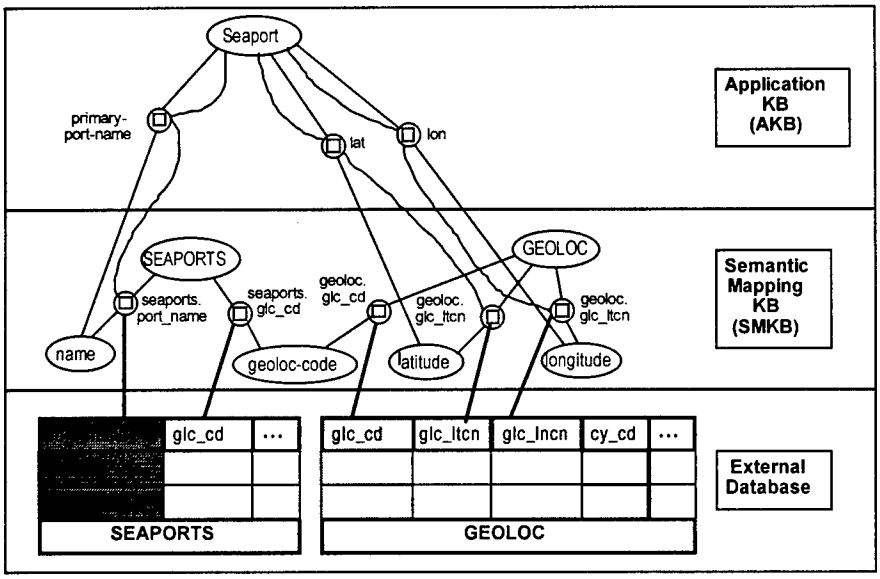


Figure 2. LIM Knowledge Base Architecture

kind of identification number.

In addition to modification of role value restrictions, it is usually desirable to represent the structural semantics of the domain more closely than is possible in the relational model. Such restructuring may be specified in the AKB by defining View-Concepts and mapping their roles to those of SMKB concepts. When such semantic restructuring takes place, it is necessary to ensure that the proposed structures are retrievable, and — if desired — updatable. ***Retrievability*** requires that all participating tables can be joined, and that sufficient information (e.g., keys) is preserved in the semantic representation to permit unambiguous access of all necessary tuples. ***Updatability*** requires that all key, index, and non-null columns in the DB tables underlying a View-Concept in the AKB are included in the View-Concept.

Retrievability of a View-Concept is assured via a mixed-initiative dialog, in which the system computes all semantically meaningful ways of joining all of the DB tables required for the construction of a View-Concept; if there is more than one such alternative, LIM presents them for selection by the user. It is not possible for the system to compute join paths without user intervention, since any given pair of tables might be joinable in several ways, not all of which are semantically equivalent.

Updatability of View-Concepts, when required, is checked automatically by the system. If any necessary information is unavailable in the view, the system identifies the missing information, and suggests that it be added to the model.

2.2.3 Query Translation

Given a LIM query, the query translation module:

- identifies variables in the query corresponding to Loom relations that are derived from the DB,
- identifies variables in the query corresponding to Loom concepts having roles derived from the DB, and
- constructs a DML query and submits it to the IDI for processing against the DB.

If the query requests the return of Loom objects, rather than just values from the DB, the DML query will select and return values in each tuple to permit generation of the appropriate Loom objects.

2.2.4 Update

The design of a DB update facility for Loom is not as straightforward as might appear at first. In this section, we will address the semantic issues involved in update; discussion of additional technical issues will be deferred to Section 2.3.

Loom instances can be constructed incrementally, by asserting the existence of the instance, and then subsequently asserting facts about it. Furthermore, classification of an instance does not take place as a result of asserting its existence, or asserting facts about it, but must be explicitly requested. In addition to the incremental nature of instance creation, certain semantic requirements must be taken into consideration: as noted in Section 2.3.3, if an instance is to be stored into a DB, an update cannot be performed unless sufficient information (e.g., values for keys and indices) is available.

While it would be possible to determine when a user considered a given instance's definition complete (i.e., when classification is requested), or to determine the point at which sufficient information has been asserted about an instance to permit storage in the DB, we saw no justification for presuming that either of these conditions should necessarily imply that the user intended to make the instance in question persistent. We therefore chose to separate instance creation from a request for storage in the DB.

When a user issues a request to store a DB-derived Loom instance, LIM verifies that it has values for all roles mapped to key- and index-columns for underlying tables, as well as for all columns for which NULL values are not permitted. If the instance contains all requisite information, the DB is modified via

the IDI. Note that a request to store an instance of a View-Concept that draws information from more than one table may result in DB operations against all such tables. Note also that the it must be decided, for each such table, whether to modify (***update***) an existing tuple, or to create (***insert***) a new tuple. Both of these issues will be discussed in Section 2.3.3.

2.2.5 Object Generation

A LIM query consists of a list of output variables to be bound, and one or more statements (in a syntax similar to that of the Loom assertional language) 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 (cf. Section 2.3.2). 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.

2.3 Technical Details

The descriptions in Section 2.2 glossed over several critical technical issues in the design, implementation, and operation of LIM. Among these are caching, data inference, complexities in update, and Meta-View-Concepts.

2.3.1 Caching

LIM uses three different caching schemes, for two purposes. The first purpose is the conventional one of improving performance; the second is related to preserving referential integrity in an Object-Oriented system. For improved performance, LIM can make use of the IDIís ***results cache***: the IDI stores the result of every query it processes, indexed under a canonicalized version of the IDIís internal representation of the query; if a subsequent query is translated to the same canonical form, *modulo* variable names, the result from the earlier query is returned. In addition, LIM employs a similar cache based on a canonicalization of the original user query: the results of each LIM query are stored, indexed under this canonical form, and a subsequent query that is identical, *modulo* variable

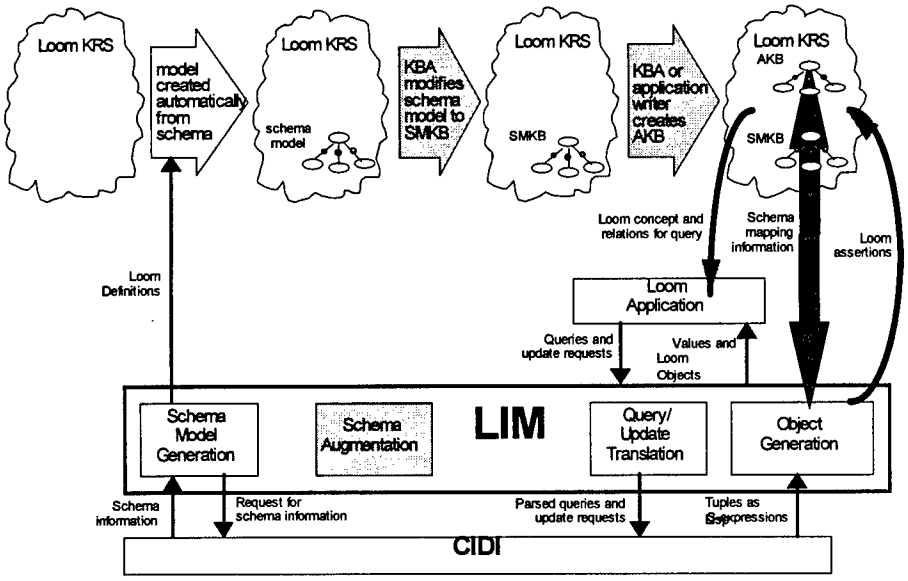


Figure 3. LIM Internal Architecture and Processing

names, will return the cached result.

Both of these can result in dramatic performance increases, because DB execution time dominates total LIM processing time for all but the largest of queries (as detailed in Section 2.3.5). A "hit" on the LIM results cache returns results almost instantaneously, since the query need not even be translated and submitted to the IDI, and a hit on the IDI results cache incurs only LIM translation time; in neither case will the DBMS be involved.

A query will hit the LIM cache only if it is virtually identical to a previous query, *modulo* variable names; on the other hand, a query might hit the IDI cache even if it is superficially quite dissimilar to an earlier query, provided that both translate to an identical IDI query. The extreme case of this is a query for an instance following a query for all of the instance's values: both will translate to the same IDI query, even though they look quite dissimilar at the surface syntactic level.

Use of both LIM and IDI caching is controllable both globally and at the individual query level. This is important in applications where the contents of the DB are known to be dynamic, and it cannot necessarily be presumed that a subsequent query *should* return the same results as an earlier identical query. We intend to augment the current caching schemes with "smart" caching techniques, such as subsumption-based caching, in future versions of LIM.

The caching used to assure referential integrity uses a similar mechanism, but for a completely different purpose. 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 s/he expects the *same* KB object to be returned in both cases. Since both results caches presently used by LIM are based purely on syntax, they are incapable of recognizing syntactically-distinct queries that should (semantically) return the same object.

A "smart" cache will alleviate this problem to a large degree, but it is still necessary to assure that syntactically-distinct queries referencing the same instance return the same object. 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.

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 (many hundred to several thousand objects). For this reason, and because of situations (like that of dynamic DB contents) where ABox cache checking is undesirable, it is controllable both globally and at the individual query level.

2.3.2 Data Inference

Data values in a DB not are necessarily in the form required or desired for the KB. LIM presently supports ***data value rendering***, which maps from simple data values in the DB to scalar KB types; e.g.,

```
"M" |C|Military
```

Future releases of LIM will support ***data object rendering***, which maps from simple data values in the DB to instances of concepts; e.g.,

```
"BSRL" |i|Bizerte(Geoloc)
```

where "BSRL" is the GEOLOC_CODE for the GEOLOC Bizerte.

Finally, LIM will support ***data classification***, which results in classification of instances in the KB after creation; e.g.,

```
> (db-retrieve ?ship
    (:and (Ship ?ship)
          (name ?s "Constitution")))

|i|Constitution(Frigate)
```

Here, it has been determined that the instance named "Constitution" is an instance of the concept Frigate, even though the query specified the more general concept Ship.

2.3.3 Update Issues

Update of simple View-Concepts that are mapped to single DB tables is relatively straightforward and unambiguous: if the View-Concept has roles mapped to all requisite columns of the underlying table (i.e., key-, index-, and no-nulls-columns), and has values for all roles so mapped, tuples underlying it can be updated or inserted. The principal subtlety in this case is determining whether an insert or update is appropriate. Our approach to making this determination is to keep track, for each instance retrieved from the DB, of the values actually retrieved. When an object is stored, LIM checks to see whether it was originally retrieved from the DB, and, if so, checks the current values against those recorded originally. If the object was not retrieved from the DB, the request to store it will be treated as an insert, and will succeed if there is no matching record in the DB. If the object was retrieved, and its key values have changed, the store request will be treated as an insert; otherwise, if its values have changed, the store request will be treated as an update, and if they have not, the request will be treated as a no-op.

When a View-Concept involves one or more joins, or has sub-View-Concepts (i.e., roles that are value-restricted to a View-Concept), the situation becomes more complex. In either case, the View-Concept is only considered updatable if all information required for the joins is stored in the View-Concept(s) involved; the complexity lies in the fact that LIM has no control over which of the two (or more) columns involved in the join(s) are mapped to the View-Concept, and it is typically the case that the values must be propagated to all of the relevant tables, even to columns that are not DB-mapped by the View-Concept. In such cases, LIM must use information about joins and subview-joins to propagate values from DB-mapped roles to both the SMKB roles to which they are mapped *and* those representing the columns across which joins are required.

The other subtlety introduced with joins is the fact that not all component tables will necessarily require either update or insert operations for all store requests. For example, short of the creation of a new seaport or airport, it is extremely unlikely that any new GEOLOCs (the DB table used to represent geographic locations of all kinds in the DB) would be created by any store request, and yet many application concepts in our domain contain information drawn from (i.e., have roles DB-mapped to) the GEOLOC table; it will seldom, if ever, be the case that a request to store an instance of one of these concepts should result in an update or insert to the GEOLOC table. Other, even more extreme, examples of this include tables like the one for COUNTRY-CODES, which maps arbitrary country codes to country names: it will seldom be the case that a routine update will want to add to, or modify, such tables.

2.3.4 Meta-View-Concepts

In addition to data about individual real-world objects like seaports and ships, some DBs (including those used by the ARPI) contain data about *classes* of real-world objects. Examples of this include data about the dozens of types of ships that are recognized. The need to represent this information in a domain model should be obvious; it should also be obvious that it is inappropriate — and potentially dangerous from the standpoint of data integrity — to permit this data to be manually encoded in the domain model.

Furthermore, it is highly desirable — if not absolutely critical — that information about classes of objects be accessible from the context of the objects themselves.

For this purpose, LIM supports a Meta-View-Concept facility. In Object-Oriented systems, the term ***metaclass*** is typically used to describe classes whose instances are all classes: just as a class describes the structure (i.e., slots or roles) of objects of that class, so a metaclass describes the structure of a class. In the case of ships, for example, a ship meta-class would describe the structure of something that represents a collection of similar ships — i.e., a class of ships — while a ship class would describe the attributes of something that represented an individual ship. While it is possible, and in many cases reasonable, for a metaclass and its classes to share attributes, it is typical for each to have attributes that are not meaningful for the other. For example, while it is reasonable that both ship classes and ships share some common attributes such as length and draft, it is meaningless for ship classes — i.e., instances of the ship metaclass — to have a hull-number attribute; and, while it is possible to represent the number of ships in its class in the object representing each individual ship — i.e., as an attribute of individual ships, defined in ship classes — it is clearly more appropriate to model this as an attribute of ship classes, defined at the ship metaclass.

LIM's Meta-View-Concept facility provides the ability to define meta-level View-Concepts, and to retrieve data for instances of these — i.e., View-Concepts — from the database. In addition to supporting queries about class-level concepts — say, for the number of ships in a particular class — meta-View-Concepts provide the basis for a mechanism implementing any of several varieties of defaults. For example, if an instance is missing a value for a particular attribute, it may be quite reasonable to obtain a proxy value by default from the instance's class; in the case of a particular type of ship, if the `length` is missing from one instance, for example, it may be reasonable to use the `length` value from the ship's class in its place. It may also be reasonable to "inherit" class-level values, such as the number of ships in a class, at the instances.

Extended Example

The diagram in Figure 4 identifies the relationships among the concepts Ship, Ship-Class, ship classes such as the CAPE H class and individual ships. Class level attributes are available from the SHIP-CLASS table in the ARPI database and individual ship attributes are retrieved from the SHIP table. This extended example explains the detailed information available in the KRSL definitions of each of these and depicted in the diagram.

Starting with the SDO definition of Ship:

```
(lim:def-view-concept ship
      :is-primitive (:and vehicle
            (:all ship-beam quantity)
            (:all ship-height quantity)
            (:all ship-max-draft number)
            (:all ship-min-draft number)
            (:all ship-max-speed quantity)
            (:all ship-service-speed quantity)
            (:all first-year-built calendar-date)
            (:all fleet-name string))
      :characteristics :clos-class)
```

we define sub-classes of Ship, for example:

```
(defconcept CAPE-H-CLASS-SHIP :is-primitive ship)
```

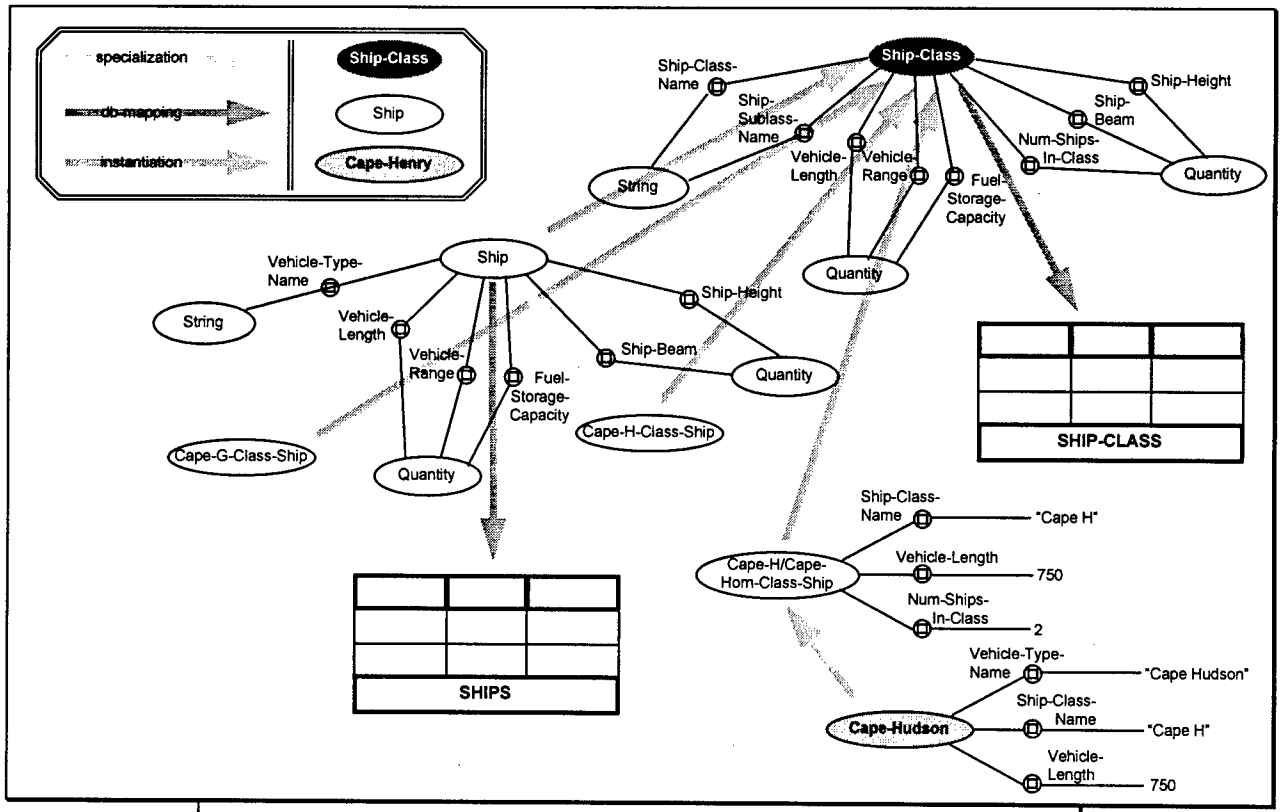


Figure 4. Metaview concepts define meta-class relationships found in databases. This example uses the two kinds of Ship information found in the Planning Initiative databases. The text explains the components of the model and how they are used.

```
(defconcept CAPE-H/CAPE-HENRY-CLASS-SHIP :is-primitive CAPE-H-CLASS-ship)

(defconcept CAPE-H/CAPE-HORN-CLASS-SHIP :is-primitive CAPE-H-CLASS-ship)
```

We then define the Meta-View-Concept for Ship-Classes:

```
(defconcept Ship-Class
      :is-primitive
           (:and Meta-View-Concept
           (:the Sh-Cl-Class-Name String) (:the Sh-Cl-Subclass-Name String)
           (:the Sh-Cl-Add-Container-Space String)(:the Sh-Cl-Class String)
           (:the Sh-Cl-Total-Sq-Ft Number)(:the Sh-Cl-Refrig-Sq-Ft Number)
           (:the Sh-Cl-Ship-Type-Pff Number)(:the Sh-Cl-Fixed-Containers Number)
           (:the Sh-Cl-Barges Number)(:the Sh-Cl-Year-Of-First-Ship Number)
           (:the Sh-Cl-Fleet String)(:the Sh-Cl-Length Number)
           (:the Sh-Cl-Beam Number)(:the Sh-Cl-Height Number)
           (:the Sh-Cl-Max-Draft Number)(:the Sh-Cl-Min-Draft Number)
           (:the Sh-Cl-Max-Speed Number)(:the Sh-Cl-Service-Speed Number)
           (:the Sh-Cl-Range Number)(:the Sh-Cl-Fuel-Cap Number)
            ..........
           (:the Sh-Cl-Seabarge-Elevators Number)(:the Sh-Cl-Pol-Cap Number))
      )
```

and map the roles of interest to the SHIP_CLASS table, in which class-level Ship information is stored:

```
(def-db-mapping ship-class Sh-Cl-Class-Name                    Ship_Class.Sh_Class)
(def-db-mapping ship-class Sh-Cl-Subclass-Name                 Ship_Class.Sh_Subclass)
(def-db-mapping ship-class Sh-Cl-Sht-Cd                        Ship_Class.Sht_Cd)
(def-db-mapping ship-class Sh-Cl-Num-Ships-In-Class
                                                        Ship_Class.Num_Ships_In_Class)
                                o
                                o
                                o
```

Finally, we declare the association between the concepts Ship and Ship-Class, and describe the roles upon whose values LIM should base the determination of the correct sub-concept of Ship to load from the DB. The mapping indicates the particular DB value which licenses an inference to a particular concept in the Loom knowledge base.

```
(declare-meta-view-concept 'Ship-Class 'Ship
  :concept-mapping-roles '(Sh-Cl-Class-Name Sh-Cl-Subclass-Name)
  :concept-role-value-mapping
   '((("ADM WM M CALLAGHAN" NIL)                ADM-WM-M-CALLAGHAN-CLASS-SHIP)
       ("AIDE" NIL)                             AIDE-CLASS-SHIP)
       ("AMBASSADOR" NIL)                       AMBASSADOR-CLASS-SHIP)
       ("AMERICAN EAGLE" NIL)                   AMERICAN-EAGLE-CLASS-SHIP)
       ("AUSTRAL LIGHTNING" NIL)                AUSTRAL-LIGHTNING-CLASS-SHIP)
       ("BANNER" NIL)                           BANNER-CLASS-SHIP)
       ("BAYAMON" NIL)                          BAYAMON-CLASS-SHIP)
       ("CAPE A CLASS" NIL)                     CAPE-A-CLASS-SHIP)
       ("CAPE B CLASS" NIL)                     CAPE-B-CLASS-SHIP)
       ("CAPE C CLASS" NIL)                     CAPE-C-CLASS-SHIP)
       ("CAPE D CLASS" NIL)                     CAPE-D-CLASS-SHIP)
       ("CAPE E CLASS" NIL)                     CAPE-E-CLASS-SHIP)
       ("CAPE F CLASS" "GREEN VALLEY")          CAPE-F/GREEN-VALLEY-CLASS-SHIP)
       ("CAPE F CLASS" "ROBERT E LEE")          CAPE-F/ROBERT-E-LEE-CLASS-SHIP)
       ("CAPE F CLASS" NIL)                     CAPE-F-CLASS-SHIP)
       ("CAPE G CLASS" NIL)                     CAPE-G-CLASS-SHIP)
       ("CAPE H CLASS" "CAPE HENRY")            CAPE-H/CAPE-HENRY-CLASS-SHIP)
       ("CAPE H CLASS" "CAPE HORN")             CAPE-H/CAPE-HORN-CLASS-SHIP)
       ("CAPE I CLASS" NIL)                     CAPE-I-CLASS-SHIP)
       ("CAPE J CLASS" NIL)                     CAPE-J-CLASS-SHIP)
       ("CAPE M CLASS" NIL)                     CAPE-M-CLASS-SHIP)
        ...........
       ("T-ACS 1-3" NIL)                        T-ACS-1-3-CLASS-SHIP)
       ("WASHINGTON" NIL)                       WASHINGTON-CLASS-SHIP))
)
```

At this stage, the definitions of all subconcepts of Ship have no information asserted about them:

```
> (pc CAPE-H-CLASS-SHIP)

(defconcept Cape-H-Class-Ship
      :is-primitive (:and Sdo-Kb^Ship
           (:all Sdo-Kb^Ship-Beam Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Height Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Max-Draft Loom::Upper-Structure-Kb^Number)
           (:all Sdo-Kb^Ship-Min-Draft Loom::Upper-Structure-Kb^Number)
           (:all Sdo-Kb^Ship-Max-Speed Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Service-Speed Sdo-Kb^Quantity)
           (:all Sdo-Kb^First-Year-Built Sdo-Kb^Calendar-Date)
           (:all Sdo-Kb^Fleet-Name Loom::Upper-Structure-Kb^String)
           (:all Sdo-Kb^Fuel-Storage-Capacity Sdo-Kb^Quantity)
           (:all Sdo-Kb^Vehicle-Range Sdo-Kb^Quantity)
           (:all Sdo-Kb^Vehicle-Length Sdo-Kb^Quantity)
           (:all Sdo-Kb^Vehicle-Description Loom::Upper-Structure-Kb^String)
           (:all Sdo-Kb^Vehicle-Type-Name Loom::Upper-Structure-Kb^String))
      :constraints
           (:and
           (:all Sdo-Kb^Vehicle-Type-Name Loom::Upper-Structure-Kb^String)
           (:all Sdo-Kb^Vehicle-Description Loom::Upper-Structure-Kb^String)
           (:all Sdo-Kb^Vehicle-Length Sdo-Kb^Quantity)
           (:all Sdo-Kb^Vehicle-Range Sdo-Kb^Quantity)
           (:all Sdo-Kb^Fuel-Storage-Capacity Sdo-Kb^Quantity)
           (:all Sdo-Kb^Fleet-Name Loom::Upper-Structure-Kb^String)
           (:all Sdo-Kb^First-Year-Built Sdo-Kb^Calendar-Date)
           (:all Sdo-Kb^Ship-Service-Speed Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Max-Speed Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Min-Draft Loom::Upper-Structure-Kb^Number)
           (:all Sdo-Kb^Ship-Max-Draft Loom::Upper-Structure-Kb^Number)
           (:all Sdo-Kb^Ship-Height Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Beam Sdo-Kb^Quantity))
      :kb Sdo-Kb)

> (pc CAPE-H/CAPE-HORN-CLASS-SHIP)

(defconcept Cape-H/Cape-Horn-Class-Ship
      :is-primitive (:and Sdo-Kb^Cape-H-Class-Ship
           (:all Sdo-Kb^Vehicle-Type-Name Loom::Upper-Structure-Kb^String)
           (:all Sdo-Kb^Vehicle-Description Loom::Upper-Structure-Kb^String)
           (:all Sdo-Kb^Vehicle-Length Sdo-Kb^Quantity)
           (:all Sdo-Kb^Vehicle-Range Sdo-Kb^Quantity)
           (:all Sdo-Kb^Fuel-Storage-Capacity Sdo-Kb^Quantity)
           (:all Sdo-Kb^Fleet-Name Loom::Upper-Structure-Kb^String)
           (:all Sdo-Kb^First-Year-Built Sdo-Kb^Calendar-Date)
           (:all Sdo-Kb^Ship-Service-Speed Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Max-Speed Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Min-Draft Loom::Upper-Structure-Kb^Number)
           (:all Sdo-Kb^Ship-Max-Draft Loom::Upper-Structure-Kb^Number)
           (:all Sdo-Kb^Ship-Height Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Beam Sdo-Kb^Quantity))
      :constraints (:and (:all Sdo-Kb^Ship-Beam Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Height Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Max-Draft Loom::Upper-Structure-Kb^Number)
           (:all Sdo-Kb^Ship-Min-Draft Loom::Upper-Structure-Kb^Number)
           (:all Sdo-Kb^Ship-Max-Speed Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Service-Speed Sdo-Kb^Quantity)
           (:all Sdo-Kb^First-Year-Built Sdo-Kb^Calendar-Date)
           (:all Sdo-Kb^Fleet-Name Loom::Upper-Structure-Kb^String)
           (:all Sdo-Kb^Fuel-Storage-Capacity Sdo-Kb^Quantity)
           (:all Sdo-Kb^Vehicle-Range Sdo-Kb^Quantity)
           (:all Sdo-Kb^Vehicle-Length Sdo-Kb^Quantity)
           (:all Sdo-Kb^Vehicle-Description Loom::Upper-Structure-Kb^String)
           (:all Sdo-Kb^Vehicle-Type-Name Loom::Upper-Structure-Kb^String))
      :kb Sdo-Kb)
```

Now, with a single retrieval request, we load all sub-concepts of Ship from the DB:

```
> (db-retrieve ?x (ship-class ?x))

(|C|AUSTRAL-LIGHTNING-CLASS-SHIP |C|PRIDE-CLASS-SHIP |C|ADM-WM-M-CALLAGHAN-CLASS-SHIP |C|CAPE-
    J-CLASS-SHIP |C|GREEN-WAVE-CLASS-SHIP |C|CAPE-G-CLASS-SHIP |C|CAPE-M-CLASS-SHIP |C|CAPE-F-
    CLASS-SHIP |C|CAPE-F/GREEN-VALLEY-CLASS-SHIP |C|CAPE-F/ROBERT-E-LEE-CLASS-SHIP |C|COMET-
    CLASS-SHIP |C|AIDE-CLASS-SHIP |C|BANNER-CLASS-SHIP |C|CAPE-A-CLASS-SHIP |C|CAPE-C-CLASS-
    SHIP |C|GULF-TRADER-CLASS-SHIP |C|T-ACS-1-3-CLASS-SHIP |C|METEOR-CLASS-SHIP |C|CAPE-B-
    CLASS-SHIP |C|WASHINGTON-CLASS-SHIP |C|DEL-MONTE-CLASS-SHIP |C|PONCE-CLASS-SHIP
    |C|BAYAMON-CLASS-SHIP |C|LYKES-CLASS-SHIP |C|CAPE-E-CLASS-SHIP |C|CAPE-D-CLASS-SHIP
    |C|FAST-SEALIFT/ALGOL-CLASS-SHIP |C|FAST-SEALIFT/CAPELLA-CLASS-SHIP |C|FAST-
    SEALIFT/ALTAIR-CLASS-SHIP |C|LYRA-CLASS-SHIP |C|JAMES-MCHENRY-CLASS-SHIP |C|CAPE-I-CLASS-
    SHIP |C|CAPE-H/CAPE-HENRY-CLASS-SHIP |C|CAPE-H/CAPE-HORN-CLASS-SHIP |C|MPS-MAERSK-CLASS-
    SHIP |C|AMBASSADOR-CLASS-SHIP |C|MAERSK-CONSTELLATION-CLASS-SHIP |C|AMERICAN-EAGLE-CLASS-
    SHIP |C|MPS-WATERMAN-CLASS-SHIP |C|MPS-AMSEA-CLASS-SHIP |C|LOGISTICS-SUPPORT-VESSEL-CLASS-
    SHIP)
```

Examining the concept CAPE-H-CLASS-SHIP, we find that it still has no assertions:

```
> (pc CAPE-H-CLASS-SHIP)

(defconcept Cape-H-Class-Ship
      :is-primitive (:and Sdo-Kb^Ship
           (:all Sdo-Kb^Ship-Beam Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Height Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Max-Draft Loom::Upper-Structure-Kb^Number)
           (:all Sdo-Kb^Ship-Min-Draft Loom::Upper-Structure-Kb^Number)
           (:all Sdo-Kb^Ship-Max-Speed Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Service-Speed Sdo-Kb^Quantity)
           (:all Sdo-Kb^First-Year-Built Sdo-Kb^Calendar-Date)
           (:all Sdo-Kb^Fleet-Name Loom::Upper-Structure-Kb^String)
           (:all Sdo-Kb^Fuel-Storage-Capacity Sdo-Kb^Quantity)
           (:all Sdo-Kb^Vehicle-Range Sdo-Kb^Quantity)
           (:all Sdo-Kb^Vehicle-Length Sdo-Kb^Quantity)
           (:all Sdo-Kb^Vehicle-Description Loom::Upper-Structure-Kb^String)
           (:all Sdo-Kb^Vehicle-Type-Name Loom::Upper-Structure-Kb^String))
      :constraints (:and (:all Sdo-Kb^Vehicle-Type-Name Loom::Upper-Structure-Kb^String)
           (:all Sdo-Kb^Vehicle-Description Loom::Upper-Structure-Kb^String)
           (:all Sdo-Kb^Vehicle-Length Sdo-Kb^Quantity)
           (:all Sdo-Kb^Vehicle-Range Sdo-Kb^Quantity)
           (:all Sdo-Kb^Fuel-Storage-Capacity Sdo-Kb^Quantity)
           (:all Sdo-Kb^Fleet-Name Loom::Upper-Structure-Kb^String)
           (:all Sdo-Kb^First-Year-Built Sdo-Kb^Calendar-Date)
           (:all Sdo-Kb^Ship-Service-Speed Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Max-Speed Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Min-Draft Loom::Upper-Structure-Kb^Number)
           (:all Sdo-Kb^Ship-Max-Draft Loom::Upper-Structure-Kb^Number)
           (:all Sdo-Kb^Ship-Height Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Beam Sdo-Kb^Quantity))
      :kb Sdo-Kb)
```

This is because there is no information in the DB for CAPE-H-CLASS-SHIP — it is a construct we introduced into the KB as a modeling convenience; CAPE-H/CAPE-HORN-CLASS-SHIP, on the other hand, corresponds to a SHIP_CLASS in the DB, and so has values asserted for it:

```
> (pc CAPE-H/CAPE-HORN-CLASS-SHIP)
```

```
(defconcept Cape-H/Cape-Horn-Class-Ship
      :is-primitive (:and Sdo-Kb^Cape-H-Class-Ship
           (:all Sdo-Kb^Vehicle-Type-Name Loom::Upper-Structure-Kb^String)
           (:all Sdo-Kb^Vehicle-Description Loom::Upper-Structure-Kb^String)
           (:all Sdo-Kb^Vehicle-Length Sdo-Kb^Quantity)
           (:all Sdo-Kb^Vehicle-Range Sdo-Kb^Quantity)
           (:all Sdo-Kb^Fuel-Storage-Capacity Sdo-Kb^Quantity)
           (:all Sdo-Kb^Fleet-Name Loom::Upper-Structure-Kb^String)
           (:all Sdo-Kb^First-Year-Built Sdo-Kb^Calendar-Date)
           (:all Sdo-Kb^Ship-Service-Speed Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Max-Speed Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Min-Draft Loom::Upper-Structure-Kb^Number)
           (:all Sdo-Kb^Ship-Max-Draft Loom::Upper-Structure-Kb^Number)
           (:all Sdo-Kb^Ship-Height Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Beam Sdo-Kb^Quantity))
      :constraints (:and (:all Sdo-Kb^Ship-Beam Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Height Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Max-Draft Loom::Upper-Structure-Kb^Number)
           (:all Sdo-Kb^Ship-Min-Draft Loom::Upper-Structure-Kb^Number)
           (:all Sdo-Kb^Ship-Max-Speed Sdo-Kb^Quantity)
           (:all Sdo-Kb^Ship-Service-Speed Sdo-Kb^Quantity)
           (:all Sdo-Kb^First-Year-Built Sdo-Kb^Calendar-Date)
           (:all Sdo-Kb^Fleet-Name Loom::Upper-Structure-Kb^String)
           (:all Sdo-Kb^Fuel-Storage-Capacity Sdo-Kb^Quantity)
           (:all Sdo-Kb^Vehicle-Range Sdo-Kb^Quantity)
           (:all Sdo-Kb^Vehicle-Length Sdo-Kb^Quantity)
           (:all Sdo-Kb^Vehicle-Description Loom::Upper-Structure-Kb^String)
           (:all Sdo-Kb^Vehicle-Type-Name Loom::Upper-Structure-Kb^String))
      :annotations ((SH-CL-CRANES-15-LTON 0) (SH-CL-CRANES-10-LTON 0) (SH-CL-SHIP-TYPE-PFF 9)
           (SH-CL-HEIGHT 158) (SH-CL-CRANES-60-LTON 0) (SH-CL-TOTAL-SQ-FT 178948)
           (SH-CL-FLEET "RRF") (SH-CL-MIN-DRAFT 15.5) (SH-CL-SEABARGE-ELEVATORS 0)
           (SH-CL-ADD-CONTAINER-SPACE "X") (SH-CL-CLASS-NAME "CAPE H CLASS")
           (SH-CL-CARGO-CAP-LTONS 26742) (SH-CL-CRANES-7-LTON 0) (SH-CL-POL-CAP 0)
           (SH-CL-CRANES-35-LTON 1) (SH-CL-SERVICE-SPEED 17) (SH-CL-CRANES-212-LTON 0)
           (SH-CL-FUEL-CAP 3638) (SH-CL-FIXED-CONTAINERS 676) (SH-CL-RORO-CAP "R")
           (SH-CL-CRANES-50-LTON 0) (SH-CL-REFRIG-SQ-FT 0) (SH-CL-NUM-SHIPS-IN-CLASS 2)
           (SH-CL-CRANES-5-LTON 0) (SH-CL-BEAM 106) (SH-CL-SUBCLASS-NAME "CAPE HORN")
           (SH-CL-MAX-DRAFT 35.6) (SH-CL-LENGTH 750) (SH-CL-MAX-SPEED 21)
           (SH-CL-SHT-CD "1402") (SH-CL-CRANES-450-LTON 0) (SH-CL-CRANES-120-LTON 0)
           (SH-CL-RANGE 24317) (SH-CL-YEAR-OF-FIRST-SHIP 1979) (SH-CL-BARGES 0)
           (SH-CL-CRANES-25-LTON 0))
      :kb Sdo-Kb)
```

2.3.5 LIM Performance

We have defined metrics for performance evaluation and have been using them continuously throughout the development of LIM. The definitions of the major components appear in Figure 5.

Using a Force Module benchmark derived from interactions between the FMERG system and the Knowledge Server during previous TIEs, we have improved average performance significantly in LIM 1.4 relative to previous releases, and by a factor of almost 40x since the original FMERG TIE. The queries all involve the retrieval of specific 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. Chart 1below 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; additional performance charts can be found on pages 71 and 72.

We have improved LIM performance dramatically since November 1992 TIE. The most notable improvements are due to the following factors:

- Loom use
  - 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

LIM Performance Measures

Components of total execution time measured are:

Augmentation	CPU time required to add concept-derived restrictions to 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 query on DBMS server
Collection	CPU time required to accumulate results of query
Object Generation	CPU time required to post-process results, including creation of Loom instances when requested
Abox Cache	CPU time required to search Loom instance DB (ABox) to prevent creation of duplicate instances, *included in Object Generation time*
Total Execution Time	Sum of all of the above *excluding Abox Cache time*

Figure 5. Definition of LIM internal performance metrics.

- Database Interface
  - We use improved fundamental data structures within the LIM and IDI implementations
  - 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.
- General
  - We tuned fundamental data structures extensively for speed and space:
    - use of lists has been dramatically reduced, since their use tends to consume cons space, and to result in excessive search:
    - where lists must be used, care is taken to avoid excessive traversal; for example, special queue structures are used when results must be appended to the ends of lists
    - components which formerly used separate data structures have been modified to share data structures wherever possible

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 reported below are dependent on the actual system used to support Oracle as well.

We have compared critical portions of the LIM execution profile, specifically, object creation and slot filling algorithms, with those availble in a commercial product expert system shell. LIM, implemented in Common Lisp and Loom, outperforms the commercial product, one written in C. Figure 1 compares the performance of LIM with that of Nexpert Object (NO) on two query sets, showing both DB execution and Object Creation. The database and queries were slected from a set developed in another project by

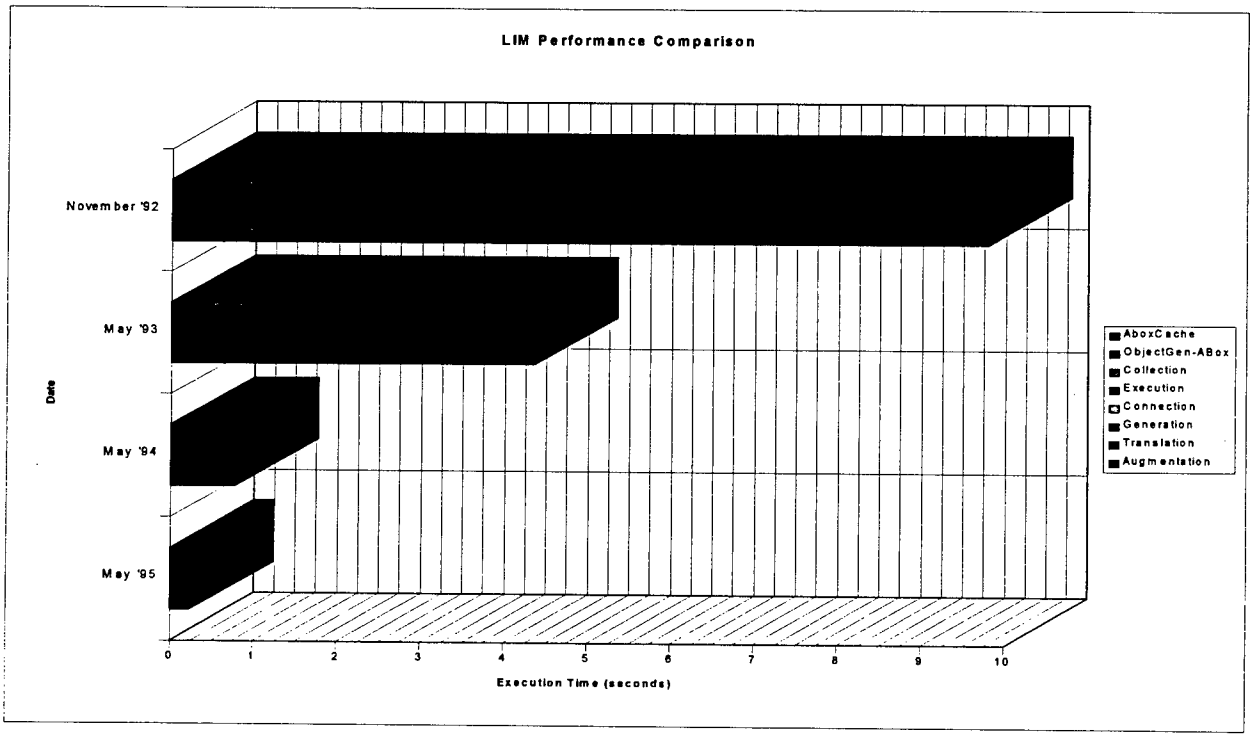


Chart 1. Comparison of LIM 1.4 performance (May '95) with previous realeases, and with original data, for FMERG force module benchmark query set.

different staff. We used LIM's ability to pick up and use a database schema to generate a skelton knowledge base in a short period of time and were executing the perfomance suite of queries in under a total of couple hours work time. While the times for LIM are approximately 25% of those for Nexpert, that is not the whole story. The LIM query set results in the creation of almost 4 times as many objects and sets almost 20 times as many slots as the NO query, and thus the differences in performance are even more substantial. For example, Figure 2 compares object creations per second of DB execution time, Figure 3 compares object creations per second of object creation time, and Figure 4 compares slots set per second of object creation time. The differences in performance range from "only" 10:1 to almost exactly 100:1. This is easily explained: the 10:1 ratio for object creations per second of DB execution time does not take into consideration the much larger size of the LIM objects, nor does the 20:1 ratio for object creations per second of object creation time. The most accurate figure is, in fact, the 100:1 figure for slot-value sets, since this accurately reflects the total amount of data being transmitted and processed. If the sum of DB execution time and object creation time is divided into the number of slot sets for LIM and NO, the overall performance advantage for LIM is on the order of 70:1.

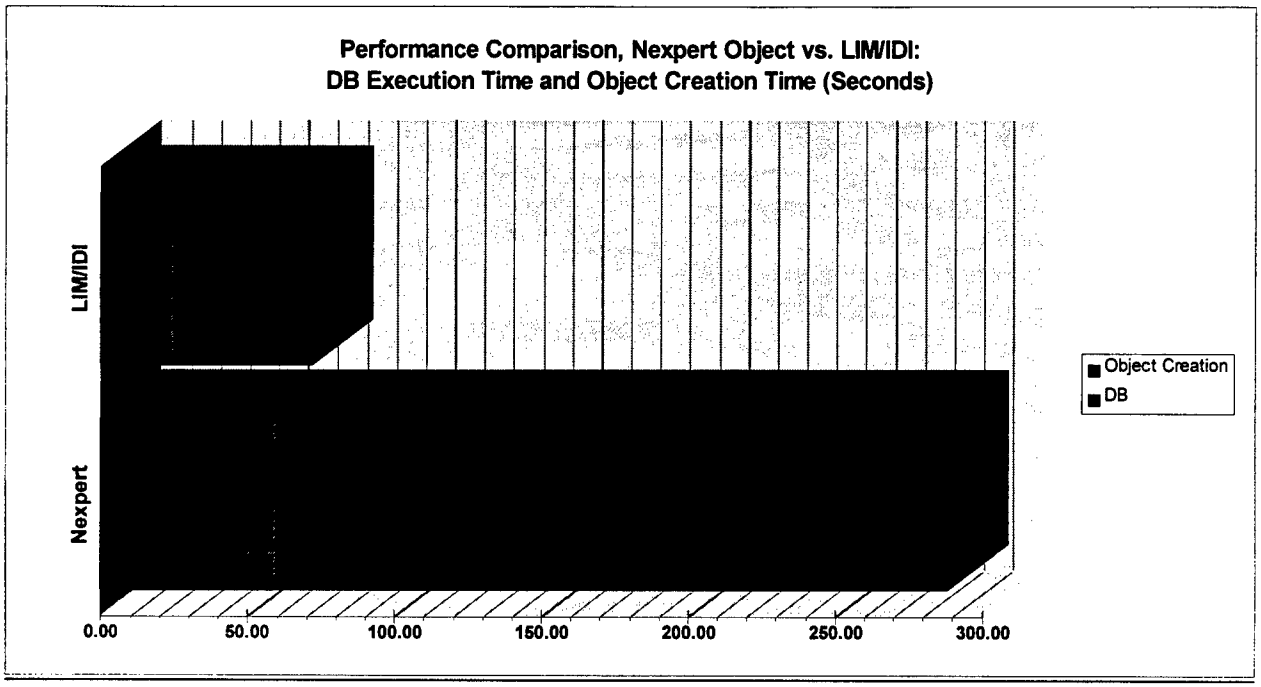


Figure 6

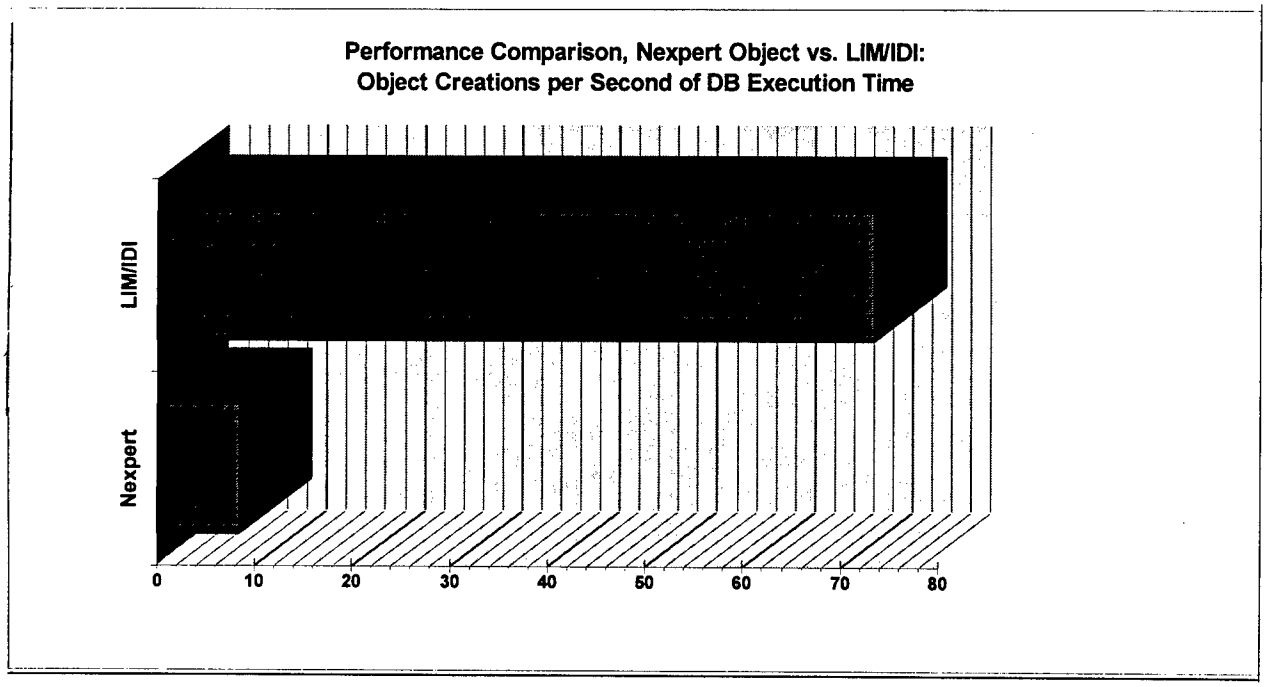


Figure 7

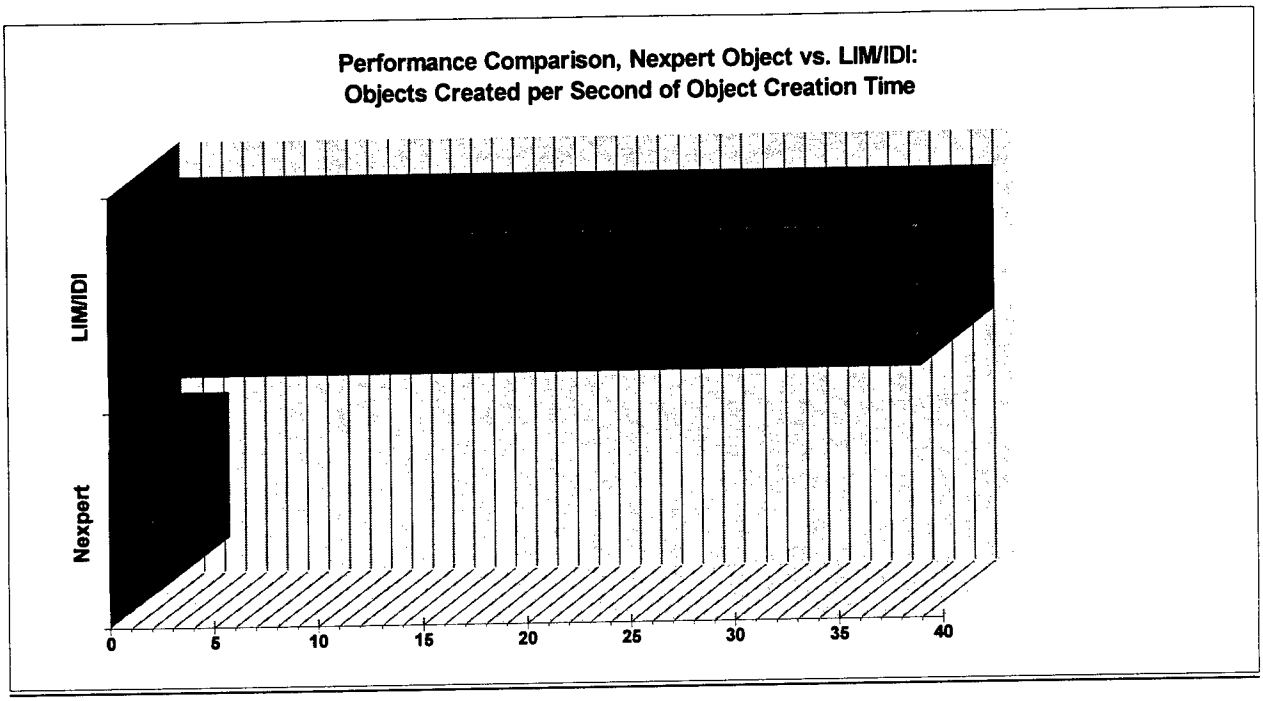


Figure 8

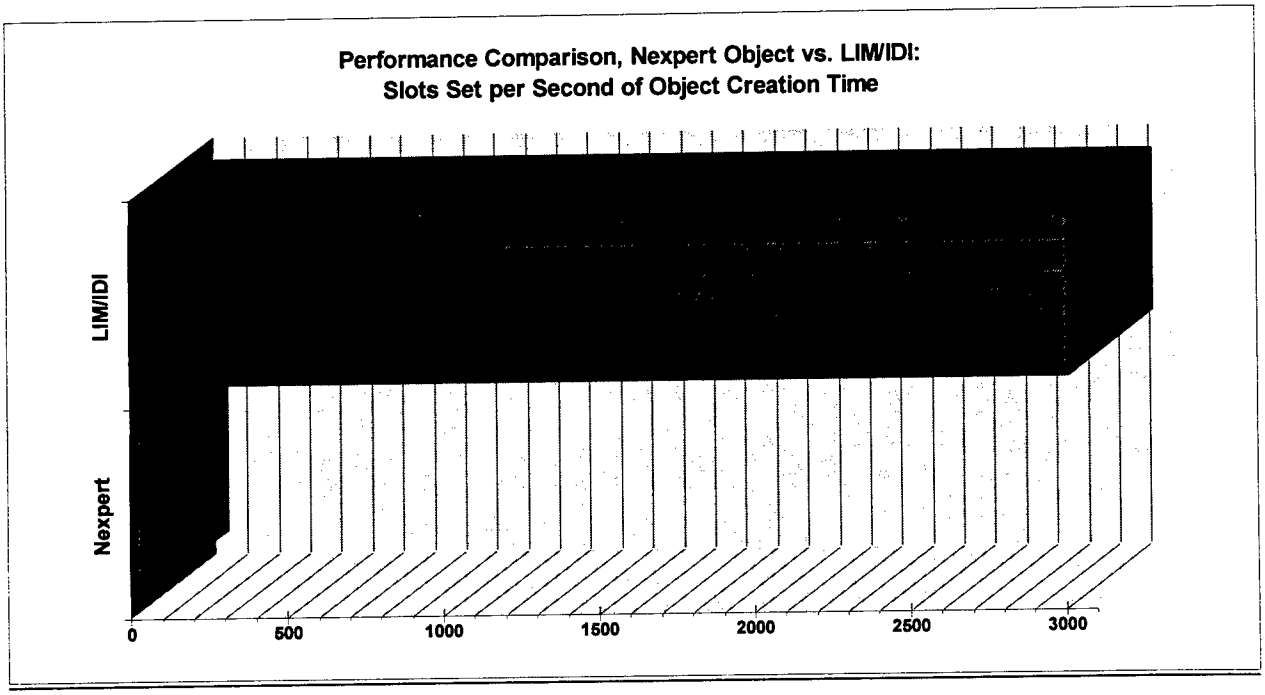


Figure 9

2.3.6 Example

To illustrate the processes described in Section 2.2, we present a small, fairly simple example. Let us presume that an application requires information about the location of various seaports. In the USTRANSCOM databases with which we are working, 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 2. Note that from a user's perspective, the SMKB and DB pre-exist, and definition of application concepts thus appears to be a top-down process; however, in order to illustrate the process of defining the mappings, we will proceed bottom-up.

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. These were created by modifying the value-restrictions in the Loom definitions automatically generated by the schema generation module. For example, the initial Loom concept definition for the portion of the GEOLOC table shown is:

```
(defconcept Geoloc
 :is-primitive
    (:and db-concept
       (:the Geoloc.Glc_cd String)
       (:the Geoloc.glc_lncn Number)
       (:the Geoloc.glc_ltcn Number)))
```

Note that role value restrictions correspond to the simple data types (e.g., string, number) that appear in relational databases. This definition is modified by the KBA to produce the SMKB definition:

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

For example, the role of Geoloc that corresponds to the column glc_cd has type string; this has been modified in the SMKB to geoloc_code. This permits LIM to infer that Seaports and Geoloc can be joined over their glc_cd roles.

Loom definitions for the semantic type hierarchies above the types used in Geoloc are:

```
(Defconcept Identifier :Is-Primitive Thing)
(Defconcept Code :Is-Primitive
          (:and String Identifier))
(Defconcept Location :Is-Primitive Code)
(Defconcept Geoloc_Code :Is-Primitive Location)
(Defconcept Measured_Qty :Is-Primitive Number)
(Defconcept Degrees :Is-Primitive Measured_Qty)
(Defconcept Latitude :Is-Primitive Degrees)
(Defconcept Longitude :Is-Primitive Degrees)
```

Finally, 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, which was created manually:

```
(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 the following declarations, which are stored as assertions in the Loom KB:

```
(def-db-mapping primary-port-name seaport seaports.port_name)
(def-db-mapping lat seaport
                  geoloc.glc_ltcn)
(def-db-mapping lon seaport
                  geoloc.glc_lncn)
```

Queries can be posed against either the SMKB or the AKB. (Note: the names used in the following examples have been changed; we have not yet obtained permission to publish the data in our test database.) 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 against the SMKB. The SQL generated by LIM and the CIDI 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:

```
("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 against the AKB. The SQL generated by LIM and the CIDI 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")))
```

2.3.7 Conclusion

We have described 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. Both are based on a verbatim model of the database; for legacy databases, this can be generated automatically from the database schema, and can be used by any knowledge-based application which would assist a knowledge base administrator in the development of the semantic mapping layer (i.e., a knowledge-based semantic schema).

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.

Finally, 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. Further, we expect that grounding the implementation in the IDI will support reasonable performance.

Our preliminary implementation includes algorithms for properly defining objects to determine retrievability and updatability, as well as retrievals against a database. In the coming year, we will be developing a more sophisticated application for the military transportation logistics domain using LIM. We expect feedback from this experience primarily to concern the completeness of the application knowledge base, and to give us valuable performance data.

3. KNOWLEDGE QUERY AND MANIPULATION LANGUAGE (KQML)

3.1 Introduction

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). The idea of an architecture where this is easy to do is quite appealing. (It is regularly mentioned in science fiction.) Many proposals for intelligent user-agents such as Knowbots [Kahn] assume the existence of this type of environment.

One type of program that would thrive in such an environment is a mediator [Wiederhold]. 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 (also known as "middleware") are becoming increasingly important as they are commonly proposed as an effective method for integrating new information systems with inflexible legacy systems.

However, networks that support "plug and play" processes, are still rare, and most distributed systems are implemented with ad hoc interfaces between their components. Many Internet resources, such as library catalog access, "finger," and menu based systems are designed to support process-to-user interaction. Those which support process-to-process communication, such as ftp or mosaic, rely on fairly primitive communication protocols. The reason for this is that there are no adequate standards to support complex communication among processes. Existing protocols, such as RPC, are insufficient for several reasons. They are not all that standard; there are currently several successful and incompatible RPC standards (ONC and DCE). They are also too low level; they do not provide high level access to information, but are intended only as "remote procedure calls."

Nor are there standard models for programming in an environment where some of the data is supplied by processes running on remote machines and some of the results are needed by other similarly distant processes. While there are many ad hoc techniques for accomplishing what is needed, it is important that standard methods are adopted as early as is reasonable in order to facilitate and encourage the use of these new architectures. It is not enough for it to be possible to communicate, it must be easy to communicate. Not only should low level communication tasks such as error checking be automatic, but using and observing protocol should be automatic as well.

KQML is a language and a protocol that supports this type of network programming specifically for knowledge-based systems or intelligent agents. It was developed by the ARPA supported Knowledge Sharing Initiative [Neches 91, Patil 92] and separately implemented by several research groups. It has been successfully used to implement a variety of information systems using different software architectures.

The Knowledge Sharing Effort

The ARPA Knowledge Sharing Effort (KSE) is a consortium to develop conventions facilitating sharing and reuse of knowledge bases and knowledge based systems. Its goal is to define, develop, and test infrastructure and supporting technology to enable participants to build much bigger and more broadly functional systems than could be achieved working alone.

The current approaches for building knowledge-based systems usually involve constructing new knowledge bases from scratch. The ability to efficiently scale up AI technology will require the sharing and reuse of existing components. This is equally true of software modules as well as conceptual knowledge. AI system developers could then focus on the creation of the specialized knowledge and reasoners new to the task at hand. New system could interoperate with existing systems, using them to perform some of its reasoning. In this way, declarative knowledge, problem solving techniques and reasoning services could all be shared among systems. The ability to build, manage and use sharable and reusable knowledge resources is thought to be a key to the realization of large-scale intelligent systems. The definition of conventions enabling sharing among collaborators is the essential first step toward these goals.

The KSE is organized around four working groups each of which is addressing a complementary problem identified in current knowledge representation technology:

- The *Interlingua Group* is concerned with translation between different representation languages, with sub-interests in translation at design time and at run-time.
- The *KRSS Group* (Knowledge Representation System Specification) is concerned with defining common constructs within families of representation languages.
- The *SRKB Group* (Shared, Reusable Knowledge Bases) is concerned with facilitating consensus on contents of sharable knowledge bases, with sub-interests in shared knowledge for particular topic areas and in topic-independent development tools/methodologies.
- The *External Interfaces Group* is concerned with run-time interactions between knowledge based systems and other modules in a run-time environment, with sub-interests in communication protocols for KB-to-KB and for KB-to-DB.

The KQML language is one of the main results which have come out of the external interfaces group of the KSE.

3.2 KQML

We could address many of the difficulties of communication between intelligent agents described in the Introduction by giving them a common language. In linguistic terms, this means that they would share a common syntax, semantics and pragmatics.

Getting information processes, especially AI processes, to share a common syntax is a major problem. There is no universally accepted language in which to represent information and queries. Languages such as KIF [Genesereth et. al. '92], extended SQL, and LOOM [MacGregor] have their supporters, but there is also a strong position that it is too early to standardize on any representation language. As a result, it is currently necessary to say that two agents can communicate with each other if they have a common representation language or use languages that are inter-translatable.

Assuming a common or translatable language, it is still necessary for communicating agents to share a framework of knowledge in order to interpret message they exchange. This is not really a shared semantics, but a shared ontology. There is not likely to be one shared ontology, but many. Shared ontologies are under development in many important application domains such as planning and scheduling, biology and medicine.

Pragmatics among computer processes includes 1) knowing who to talk with and how to find them and 2) knowing how to initiate and maintain an exchange. KQML is concerned primarily with pragmatics (and secondarily with semantics). It is a language and a set of protocols that support computer programs in identifying, connecting with and exchanging information with other programs.

3.3 KQML Protocols

There are a variety of interprocess information exchange protocols. There is the simple case of one process (a client) sending a query to another process (a server) and waiting for a reply as is shown in Figure 10. This occurs commonly when a backward-chaining reasoner retrieves information from a remote source. As it needs data, it places queries and waits for the replies before attempting any further inferences. This case includes those where the server's reply message actually contains a collection of replies.

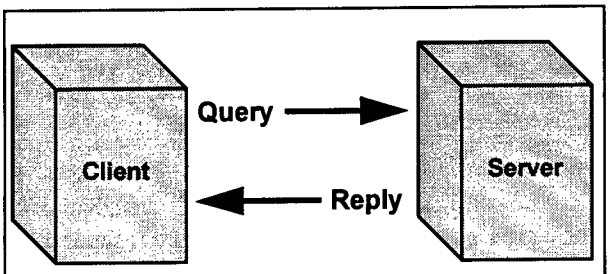


Figure 10. In this example of a *synchronous communication query*, a blocking query waits for an expected reply.

Another common case is when the server's reply is not the complete answer but a handle which allows the client to ask for the components of the reply, one at a time as shown in Figure 11. A common example of this type of exchange is a simple client querying a relational database or a reasoner which can produce a sequence of instantiations in response to a query. Although this exchange requires that the server maintain some internal state, the individual transactions are each the same as in the single reply case. I.e., each transaction is a "send-a-query / wait / receive-a-reply" exchange. We refer to these transactions as being synchronous because messages arrive at the client only when they are expected.

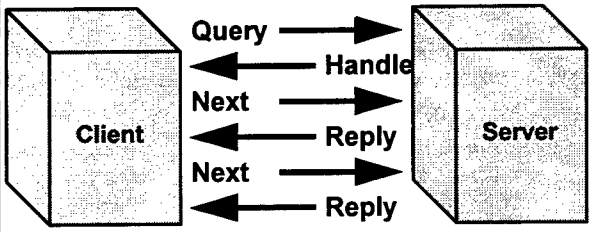


Figure 11. Remote server can maintain state by remembering the partial answer. Replies are sent individually, each at the request of the client.

A somewhat different case occurs in real-time systems (among others) where the client subscribes to a server's output and an indefinite number of replies arrive at irregular intervals in the future, as shown in Figure 12. In this case, the client does not know when each reply message will be arriving and may be busy performing some other task when they do. We refer to these transactions as being asynchronous.

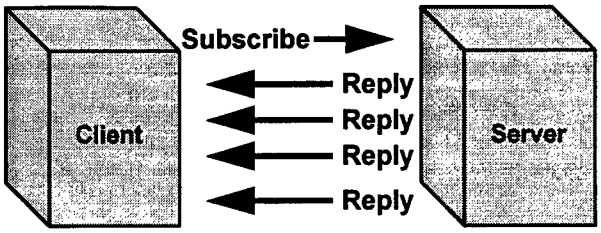


Figure 12. Asynchronous Communication Protocol - A non-blocking subscribe request results in an irregularly spaced, indeterminate number of incoming messages

There are other variations of these protocols. Messages might not be addressed to specific hosts, but broadcast to a number of them. The replies, arriving synchronously or asynchronously have to be collated and, optionally, associated with the query that they are replying to.

3.4 The KQML Language

KQML supports these protocols by making them an explicit part of the communication language. When using KQML, a software agent transmits messages composed in its own representation language, wrapped in a KQML message.

KQML is a layered language. The KQML language can be viewed as being divided into three layers: the content layer, the message layer and the communication layer. The content layer is the actual content of the message, in the programs own representation language. 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.

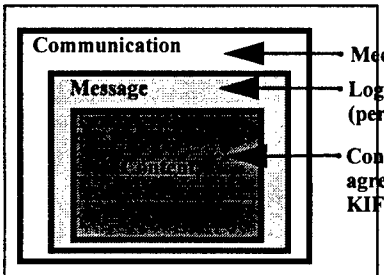


Figure 13. The KQML language can be viewed as being divided into three layers: the content layer, the message layer and the communication layer.

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.

The message layer forms the core of the language. It determines the kinds of interactions one can have with a KQML-speaking agent. The primary function of the message 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: its language, the ontology it assumes, and some type of more general

description, such as a descriptor naming a topic within the ontology. These features make it possible for KQML implementations to analyze, route and properly deliver messages even though their content is inaccessible.

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)
```

The syntax of KQML is based on a balanced parenthesis list. The initial element of the list is the performative and the remaining elements are the performative's arguments as keyword/value pairs. Because the language is relatively simple, the actual syntax is relatively unimportant and can be changed if necessary in the future. (The current syntax was selected because most of the original implementation efforts were done in Common Lisp.)

The set of KQML performatives is extensible. There is a set of reserved performatives which have a well defined meaning. This is not a required or minimal set; a KQML agent may choose to handle only a few (perhaps one or two) performatives. However, an implementation that chooses to implement one of the reserved performatives must implement it in the standard way. A community of agents may choose to use additional performatives if they agree on their interpretation and the protocol associated with each.

Some of the reserved performatives are shown in Figure 14. In addition to standard communication performatives such as ask, tell, deny, delete, and more protocol oriented performatives such as subscribe, KQML contains performatives related to the non-protocol aspects of pragmatics, such as advertise - which allows an agent to announce what kinds of asynchronous messages it is willing to handle; and recruit - which can be used to find suitable agents for particular types of messages.

For example, agent B might send the following performative to agent A:

```
(advertise
    :language KQML
:ontology K10
:content (subscribe :language KQML
:ontology K10
:content (stream-about
:language KIF
:ontology motors
:content motor1)))
```

to which agent B might responds with:

```
(subscribe :reply-with s1
:language KQML
:ontology K10
:content (stream-about
:language KIF
:ontology motors
:content motor1))
```

Agent A would then send B a stream of tell and untell performatives over time with information about motor1, as in:

```
(tell :language KIF
:ontology motors
:in-reply-to s1
:content (= (val (torque motor1) (sim-time 5))
(scalar 12 kgf))
(tell :language KIF
:ontology structures
:in-reply-to s1
:content (fastens frame12 motor1))
(untell :language KIF
:ontology motors
:in-reply-to s1
:content (= (val (torque motor1) (sim-time 5))
(scalar 12 kgf))
...
```

Basic query performatives:

evaluate, ask-if, ask-in, ask-one, ask-all

Multi-response query performatives:

stream-in, stream-all

Response performatives:

reply, sorry

Generic informational performatives:

tell, achieve, cancel, untell, unachieve

Generator performatives:

standby, ready, next, rest, discard, generator

Capability-definition performatives:

advertise, subscribe, monitor, import, export

Networking performatives:

register, unregister, forward, broadcast, route

Figure 14. There are about two dozen reserved performative names which fall into seven basic categories.

KQML Semantics. A semantic model is under development that assumes that a KQML-speaking agent has a virtual knowledge base with two separate components: an information store (i.e., "beliefs") and a goal store (i.e., "intentions"). The primitive performatives are defined in terms of their effect on these sores. A TELL(S), for example, is an assertion by the sending agent to the receiving agent that a particular sentence is in its virtual belief store. An ACHIEVE(S) is a request of the sender to the receiver to add S it its intention store.

The protocols that govern the allowable responses when an agent receives a KQML message must also be defined. These are currently defined informally in English descriptions, but work is underway to provide formal definitions in terms of a grammar using the definite clause grammar (DCG) formalism.

KQML Internal Architectures KQML was not defined by a single research group for a particular project. It was created by a committee of representatives from different projects, all of which were concerned with managing distributed implementations of systems. One was a distributed collaboration of expert systems in the planning and

scheduling domain. Another was concerned with problem decomposition and distribution in the CAD/CAM domain. A common concern was the management of a collection of cooperating processes and the simplification of the programming requirements for implementing a system of this type. However, the groups did not share a common communication architecture. As a result, KQML does not dictate a particular system architecture, and several different systems have evolved.

Our group has two implementations of KQML. One is written in Common Lisp, the other in C. Both are fully interoperable and are frequently used together.

The design of these implementations was motivated by the need to integrate a variety of preexisting expert systems into a collaborating group of processes. Most of the systems involved were never designed to operate in a communication oriented environment. The design is built around two specialized programs, a *router* and a *facilitator*, and a library of interface routines, called a *KRIL*.

KQML Routers. *Routers* are content independent message routers. Each KQML speaking software agent is associated with its own separate router process. All routers are identical; each is just an executing copy of the same program. A router handles all KQML messages going to and from its associated agent. Because each program has an associated router process, it is not necessary to make extensive changes to each program's internal organization to allow it to asynchronously receive messages from a variety of independent sources. The router provides this service for the agent and provides the agent with a single point of contact for the rest of the network. It provides both client and service functions for the application and manages multiple simultaneous connections with other agents.

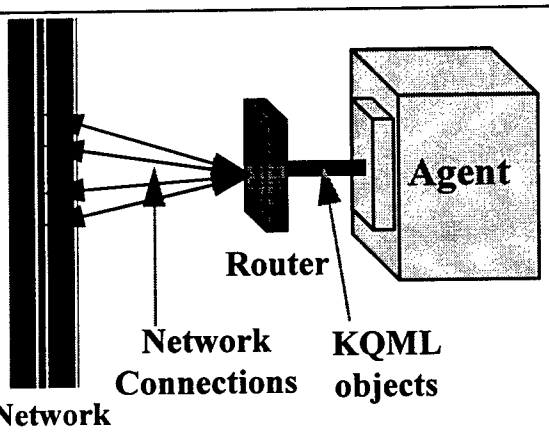


Figure 15. A router gives an application a single interface to the network, providing both client and server capabilities, managing multiple simultaneous connections, and handling some KQML interactions autonomously.

The router never looks at the content fields of the messages it handles. It relies on the KQML performatives and its arguments. If an outgoing KQML message specifies a particular Internet address, the router directs the message to it. If the message specifies a particular service, the router will attempt to find an Internet address for that service and deliver the message to it. If the message only provides a description of the content (e.g. query, :ontology "geo-domain-3", :language "Prolog", etc.) the router may attempt to find a server which can deal with the message and it will deliver it there, or it may choose to forward it to a smarter communication agent which may be willing to route it. Routers can be implemented with varying degrees of sophistication -- they can not guarantee to deliver all messages.

KQML Facilitators. To deliver messages that are incompletely addressed, routers rely on *facilitators*. A facilitator is a network application which provides useful network services. It maintains a registry of service names;it will forward messages on request to named services. It may provide matchmaking services between information providers and consumers. Facilitators are actual network software agents; they have their own KQML routers to handle their traffic and they deal exclusively in KQML messages. There is typically one facilitator for each local group of agents. This can translate into one facilitator per local site or one per project; there may be multiple local facilitators to provide redundancy.

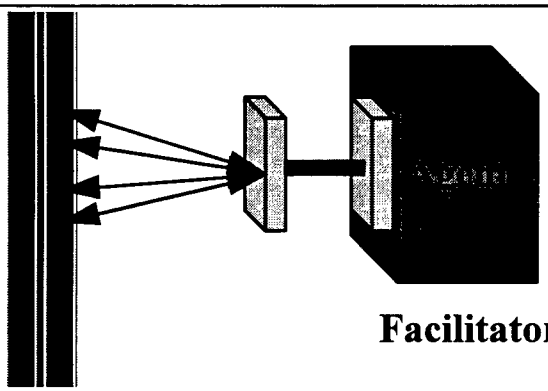


Figure 16. A Communication Facilitator is an agent that performs various useful services, e.g. maintaining a registry of service names, forwarding messages to named services, routing messages based on content, providing "matchmaking" between information providers and clients, and providing mediation and translation services.

When each application starts up, its router announces itself to the local facilitator so that it is registered in the local database. When the application exits, the router sends another KQML message to the facilitator, removing the application from the facilitator's database. In this way applications can find each other without there having to be a hand maintained list of local services.

KQML KRILs. Since the router is a separate process from the application, it is necessary to have a programming interface between the application and the router. This interface is called a KRIL (KQML Router Interface Library). While the router is a separate process, with no understanding of the content field of the KQML message, the KRIL is embedded in the application and has access to the application's tools for analyzing the content. While there is only one piece of router code, which is instantiated for each process, there can be various KRILs, one for each application type or one for each application language. The general goal of the KRIL is to make access to the router as simple as possible for the programmer.

To this end, a KRIL can be as tightly embedded in the application, or even the application's programming language, as is desirable. For example, an early implementation of KQML featured a KRIL for the Prolog language which had only a simple declarative interface for the programmer. During the operation of the Prolog interpreter, whenever the Prolog database was searched for predicates, the KRIL would intercept the search; determine if the desired predicates were actually being supplied by a remote agent; formulate and pose an appropriate KQML query; and return the replies to the Prolog interpreter as though they were recovered from the internal database. The Prolog program itself contained no mention of the distributed processing going on except for the declaration of which predicates were to be treated as remote predicates. Figure 19 shows an example of this together with a facilitation agent which provides a central content-based routing service.

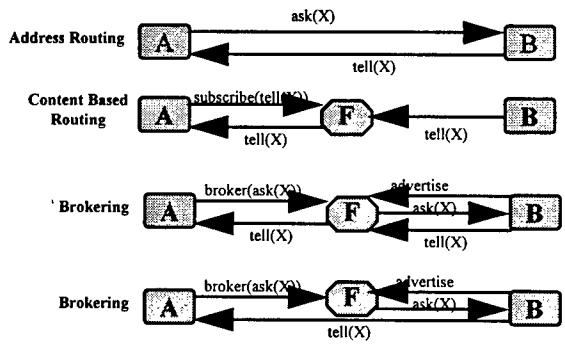


Figure 17. Facilitators are agents that deal in knowledge about the information services and requirements of other agents. They can offer services such as forwarding, brokering, recruiting and content-based routing.

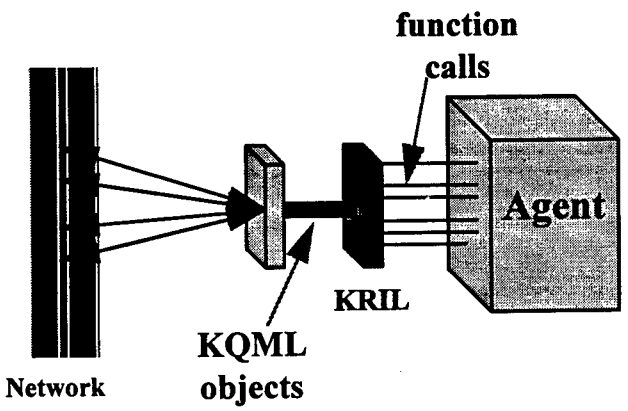


Figure 18. The KRIL is part of the application and has access to its internals. It provides internal access points to which the router delivers incoming messages, analyzes outgoing messages for appropriate domain tagging and routing, and provides application specific interface and procedures for communication access.

It is not necessary to completely embed the KRIL in the application's programming language. A simple KRIL generally provides two programmatic entries. For initiating a transaction there is a **send-kqml-message** function. This accepts a message content and as much information about the message and its destination as can be provided and returns either the remote agent's reply (if the message transmission is synchronous and the process blocks until a reply is received) or a simple code signifying the message was sent. For handling incoming asynchronous messages, there is usually a **declare-message-handler** function. This allows the application programmer to declare which functions should be invoked when messages arrive. Depending on the KRILs capabilities, the incoming messages can be sorted according to *performative*, or *topic*, or other features, and routed to different message handling functions.

In addition to these programming interfaces, KRILs accept different types of declarations which allow them to register their application with local facilitators and contact remote agents to advise them that they are interested in receiving data from them. Our group has implemented a variety of experimental KRILs, for Common Lisp, C, Prolog, Mosaic, SQL, and other tools.

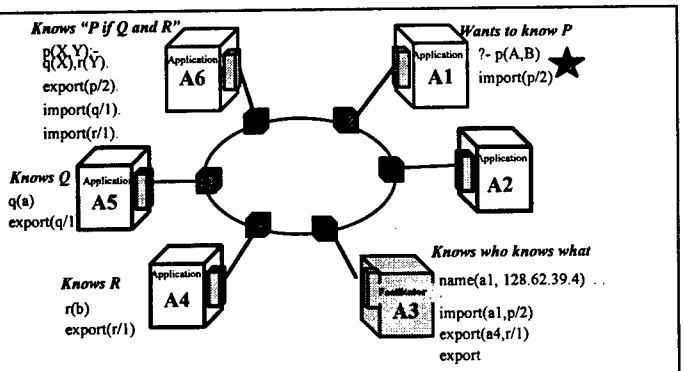


Figure 19. This example shows the use of a facilitator to do content-based routing allowing a set of Prolog-based agents to work together to prove goals.

3.5 KQML Performance

We have developed a simple performance model for KQML communication which has allowed us to analyze the efficiency of communication and to identify and eliminate bottlenecks by tuning the software and adding additional capabilities. For example, various compression enhancements have been added which cut the communication costs by reducing the message sizes and also by eliminating a substantial fraction of symbol lookup and string duplication.

One of the serious system integration comments from BBN and ISX has concerned the execution performance of knowledge servers. In particular, we have noticed considerable delays in communication which have not been accounted for by normal network traffic and latency. In an attempt to understand this issue, we have developed a performance model for communication among intelligent agents. The performance model defines metrics from the standpoint of a sender and receiver agent (see Figure 20). Performance is measured both with respect to the apparent elapsed time as observed from the sender but also from the perspective of the receiver. For example, in a simple test, the sender sends a message to the receiver and subsequently receives a reply. For the sender, there is write time to actually write the message on the communication channel, wait time while the receiver actually processes the message and read time once the reply has been sent and is available to be read. Similarly, for the receiver, there is the time to actually read the message from the sender, process or execute the message and the time to actually write the message back to the receiver. In a simple test where the sender sent a short coded message to the receiver for the receiver to reply with an already constructed message, we observed excessive elapsed times for large responses. We derived our sample set of messages from the set of messages passed among the knowledge server and an agent in the Common Prototyping Environment that we had collected previously. It appeared the read time of the sender was 2-3 times greater than the write time for the receiver. Lucid Common Lisp's implementation of TCP/IP streams uses the Lucid multi-tasking environment, and, since the scheduling of tasks is non-preemptive, the task which handled the sender's reading of the response was getting every third time slice of the Lucid task scheduler. The specific times observed were due to the setting of the time slice interval, i.e., the sender was only reading one half to one third of the time and was otherwise not occupied. We have begun developing operational strategies to cope with the Lucid scheduler and have introduced a communications switch which manipulates the time slice setting so that one can choose rapid I/O with almost no time delay with near lock out of other processing or choose normal I/O with moderate time delay.

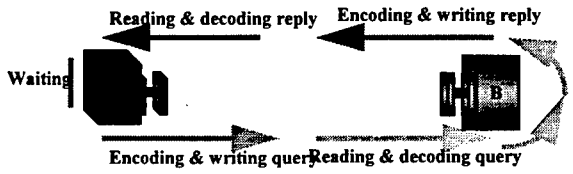


Figure 20. A simple performance model for KQML communication has allowed us to analyze the efficiency of KQML-based communication and to eliminate bottlenecks.

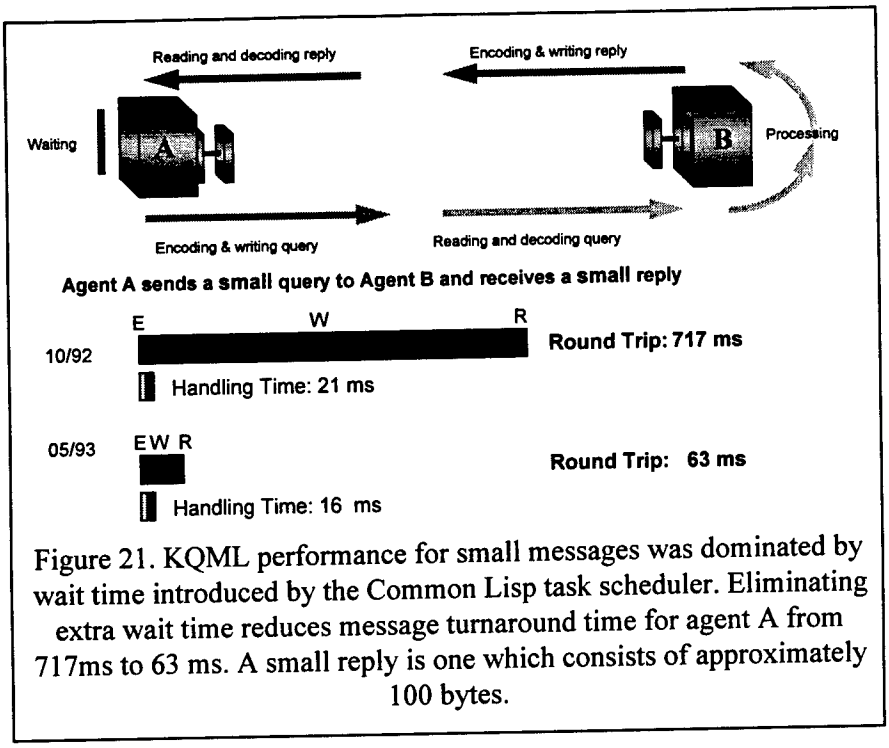


Figure 21. KQML performance for small messages was dominated by wait time introduced by the Common Lisp task scheduler. Eliminating extra wait time reduces message turnaround time for agent A from 717ms to 63 ms. A small reply is one which consists of approximately 100 bytes.

Figure 21 shows the case for small messages in which the sender has an excessive wait time even though both the sender and receiver spend little time actually doing and I/O. The wait time of the sender is nearly determined by the 333ms time slice of the Common Lisp task scheduler. By eliminating the excessive time slicing, we reduced the wait time considerably. In Figure 22, for larger messages which cannot be all read within on time slice, the observation is that the sender spends more time reading what the receiver sent by about a factor of 1.5-2.

We have tested a direct compression technique for KRSL query and object instance response forms. The compression technique is a simple dictionary coding scheme for S-expressions, it is similar to "hash consing" where a given sub-expression appears exactly once in the input stream and is otherwise referenced from the code dictionary. Initial experiments indicate that the redundancies present in KRSL object forms for force modules, e.g., repetition of slot names filled by the same value, occur less

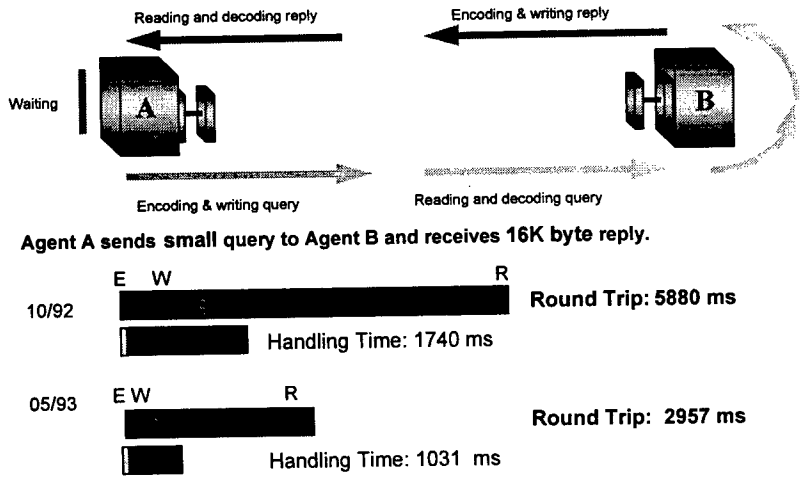


Figure 22. Performance for larger messages is dominated by Common Lisp I/O processing which accounts for almost all of the read time. For uncompressed messages, performance has been improved by up to a factor of 2. Compression further reduces the read time for the agent.

frequently than expected. A simple coding technique which codes only Lisp symbols seems to be most effective in terms of encode-decode time and space performance. This reduces the number of times the Lisp reader does a symbol lookup in a given package to one. This is appears to be the significant component of read-write performance for handling KRSL forms. We believe that this approach will very quickly lead to performance that BBN achieved special purpose communication code within their CRONUS product. Our version is relatively straight forward Lisp code that can be shared with the Planning Initiative. Initial performance measurements indicate between a 2 and 5 fold improvement. As the compression technique both reduces the message size for highly redundant messages and eliminates some of the excess Common Lisp I/O time, the actual performance improvement is subject to the structure and content of the messages. In our test set, we have used KRSL messages collected from actual message sets involving Force Modules. Since they are highly redundant, the results for compression are about as good as one would expect to see.

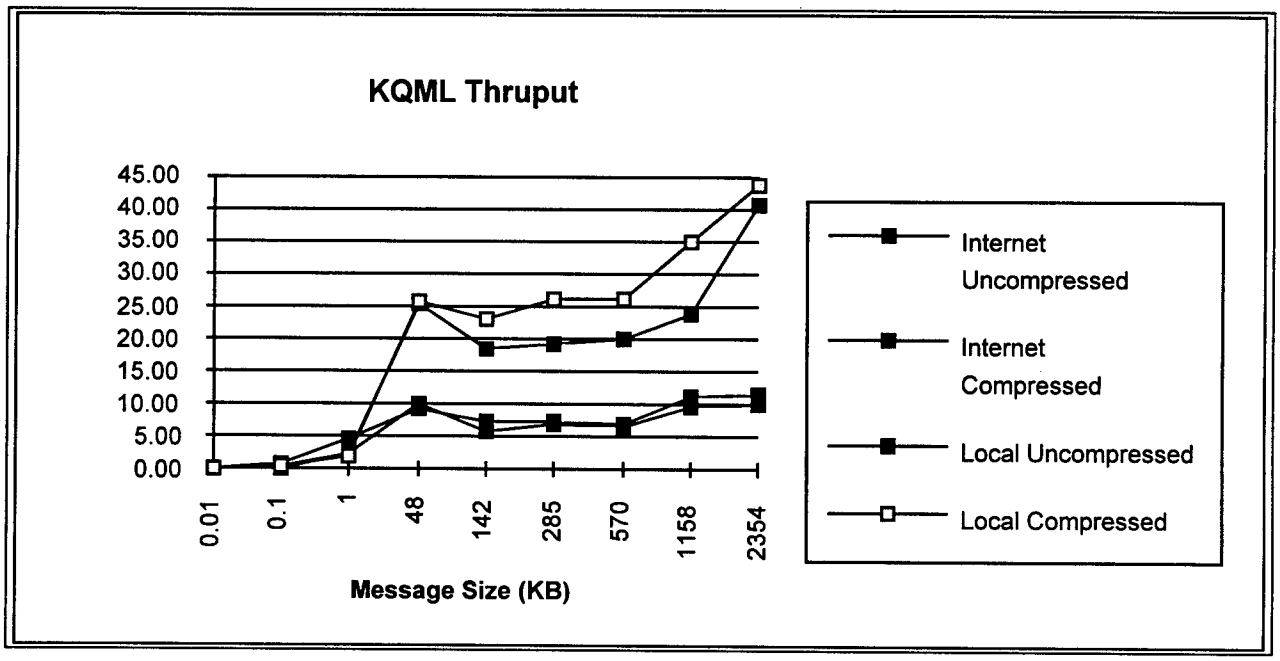


Figure 23. KQML throughput for local networks and the internet. Compression leads to sustainable rates of 25KB/sec for messages of 150KB to over 2MB. Uncompressed messaging is dominated by Common Lisp i/o time.

3.6 Lessons Learned

This Section has described KQML -- a language and associated protocol by which intelligent software agents can communicate to share information and knowledge. We believe that KQML, or something very much like it, will be important in building the distributed agent-oriented information systems of the future. One must ask how this work is to be differentiated from the work in two closely related areas -- *distributed systems* and *distributed AI.*

KQML and Distributed Systems. There has been considerable good work involving distributed systems theory and practice in the past decades. This is also true for this work as it relates to distributed information systems such as distributed databases and distributed object systems. We feel that what KQML offers is an abstraction of an information agent (provider or consumer) at a higher level that is typical in other areas of Computer Science. In particular, KQML assumes a model of an agent as a knowledge-based system (KBS). Although this will not seem to be surprising or profound in our AI community, it is a significant advance (we hope!) for the general CS community. The KBS model easily subsumes a broad range of commonly used information agent models in use today, including database management systems, hypertext systems, server-oriented software (e.g. finger demons, mail servers, HTML servers, etc), simulations, etc. Such systems can usually be modeled as having two virtual knowledge bases -- one representing the agent's information store (i.e., beliefs) and the other representing its intentions (i.e., goals).

We hope that future standards for interchange and interoperability languages and protocols will be based on this very powerful and rich model. This will avoid the built-in limitations of more constrained models (e.g., that of a simple remote procedure call or relational database query) and also make it easier to integrate truly intelligent agents with simpler and more mundane information clients and servers.

In addition to having something to offer, KQML also has something it seeks from distributed systems work -- the right abstractions and software components to provide basic communication services. Current KQML-based systems have been built on the most common transport layers in use today -- TCP/IP and EMAIL. The real contributions that KQML makes are independent of the transport layer. We anticipate that KQML interface implementations will be based on whatever is seen as the best transport mechanism.

KQML and Distributed AI. The contribution that KQML makes to Distributed AI research is to offer a standard language and protocol that intelligent agents can use to communicate among themselves as well as with other information servers and clients. We believe that the flexibility afforded by KQML permitting agents to use whatever content language they like will make it appropriate for most DAI research. In designing KQML, our goal is to build in the primitives necessary to support all of the interesting agent architectures currently in use. If we have been successful, then KQML should serve to be a good tool for DAI research, and, if used widely, should enable greater research collaboration among DAI researchers.

KQML and the Future. The ideas which underlie the evolving design of KQML are currently being explored through experimental prototype systems which are being used to support several testbeds in such areas as concurrent engineering [Cutkowski, McGuire, Tenenbaum, Kuokka], intelligent design [Genesereth] and intelligent planning and scheduling. We have also used KQML, for example, to support the interchange of knowledge among a planner, a plan simulator, a plan editor and a knowledge server, which is the repository for the shared ontology and access point to common databases through the *Intelligent Database Interface* [McKay, Pastor].

We have developed a simple performance model for KQML communication which has allowed us to analyze the efficiency of communication and to identify and eliminate bottlenecks by tuning the software and adding additional capabilities. For example, various compression enhancements have been added which cut the communication costs by reducing the message sizes and also by eliminating a substantial fraction of symbol lookup and string duplication.

4. KNOWLEDGE SERVER ARCHITECTURE AND EXAMPLES

4.1 KQML as a Knowledge Server Language

4.1.1 What is an Agent Communication Language?

A wide variety of systems, languages, frameworks and standards efforts are associated with software agents; this is due in part to the vagueness of the term 'software agent.' In this section we attempt to tease apart these approaches, and show how KQML relates to each of them. We divide agent technologies into two broad categories: agent languages and coordination protocols.

The agent languages category comprises all languages that can be used to implement software agents. Virtually any programming language can be used for software agent development. One class of languages that has gained much attention lately is the so—called scripting languages, especially those designed for mobile programs. Languages like Tcl/Tk, Java, Telescript, etc., offer the advantage of a level of abstraction that seems particularly attractive for the development of software agents. We place them under the agent languages rubric, because they can be used to program software agents. We distinguish them from agent communication languages though because they are designed primarily to control processes on a single platform. To the extent that these languages contain communication primitives tailored to agent development, they are largely concerned with the transportation of a single agent from one machine to another.

In contrast, agent communication languages are designed specifically to describe and facilitate communication among two or more agents. Three broad sub—categories may be identified under the label of agent communication languages: models of human communication, theoretical frameworks, and communication languages for software agents. Human communication is traditionally modeled in terms of speech act theory. Considerable work has been done (e.g., Cohen and Lesveque, Singh) to capture the assumptions and conventions of interaction between human agents and subsequently translate them into workable paradigms for the development of their artificial counterparts. Often, such work leads to theoretical frameworks for artificial agents with human—like capabilities. Such frameworks attempt to account for all aspects of the internal state of an artificial autonomous agent, with a particular attention to how this state changes as the agent interacts (and/or communicates) with the world or with other agents. Sometimes, as in the case of Agent Oriented Programming, those frameworks may evolve into implemented software systems. In contrast, agent communication languages (ACL) are concerned strictly with the communication between such computational entities. An ACL (the sub—category that includes KQML) is more than a protocol for the exchange of data, because an attitude about what is exchanged by the agents is also communicated. An ACL may be thought as a communication protocol (or a collection of protocols) that supports many message types.

The other main class of software agent technologies is that of standards and coordination protocols. 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 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.

4.1.2 Desiderata

In this section we identify requirements for agent communication languages. We divide these requirements into seven categories: form, content, semantics, implementation, networking, environment, and reliability. We believe that an agent communication language will be valuable to the extent that it meets these requirements. At times, these requirements may be in conflict with one another. For example, a language that can be easily read by people might not be as concise as possible. It is the job of the language designer to balance these various needs.

Form. A good agent communication language should be declarative, syntactically simple, and readable by people. It should be concise, yet easy to parse and to generate. To transmit a statement of the language to another agent, the statement must pass through the bit stream of the underlying transport mechanism. Thus, the language should be linear or should be easily translated into a linear stream of characters. Because a communication language will be integrated into a wide variety of systems, its syntax should be extensible. Finally, the form should be stylistically acceptable to a variety of communities, possibly through the use of "syntactic sugar."

Content. A communication language should be layered in a way that fits well with other systems. In particular, a distinction should be made between the communication language, which expresses communicative acts, and the content language, which expresses facts about the domain. Such layering facilitates the successful integration of the language into applications while providing a conceptual framework for the understanding of the language. The language should commit to a well—defined set of communicative acts (primitives). Although this set could be extensible, a core set of primitives that captures most of our intuitions about what constitutes a communicative act irrespective of application (database, object—oriented system, knowledge base, etc.) will ensure the usability of the language by a variety of systems. The choice of the core set of primitives also affects the decision of whether to commit to a specific content language. A commitment to a content language allows a more restricted set of communicative acts, because it is then possible to carry more information at the content language level. The disadvantage of commitment to a content language is that all applications must then use the same content language; this is a heavy constraint.

Semantics. Semantics is an issue that has often been neglected during the design of communication languages. Such neglect is the direct result of the obscurity that surrounds the purpose and the desired features of communication languages. Although the semantic description of communication languages and their primitives is often limited to natural language descriptions, a well—defined semantic description is anything but a luxury. This is especially true if the communication language is intended for interaction among a diverse range of applications. Applications designers should have a shared understanding of the language, its primitives and the protocols associated with their use, and abide by that shared understanding.

The semantics of a communication language should exhibit those properties expected of the semantics of any other language. It should be grounded in theory, and it should be unambiguous. It should exhibit canonical form (similarity in meaning should lead to similarity in representation). Because a

communication language is intended for interaction that extends over time among spatially dispersed applications, location and time should be carefully addressed by the semantics. Finally, the semantic description should provide a model of communication, which will be useful for performance modeling, among other things.

Implementation. The implementation should be efficient, both in terms of speed and bandwidth utilization. It should provide a good fit with existing software technology. The interface should be easy to use; details of the networking layers that lie below the primitive communicative acts should be hidden from the user. It should be easy to integrate or build application program interfaces for a wide variety of programming languages, including procedural languages (e.g., C and Lisp), scripting languages (e.g., Tcl and Perl), object—oriented languages (e.g., Smalltalk and C++), and logic programming languages (e.g., Prolog). Finally, the language should be amenable to partial implementation, because simple agents may only need to handle a subset of the primitive communicative acts.

Networking. An agent communication language should fit well with modern networking technology. This is particularly important because some of the communication will be about concepts involving networked communications. The language should support all of the basic connection types---point—to—point, multicast and broadcast. Both synchronous and asynchronous connections should be supported. The language should contain a rich enough set of primitives that it can serve as a substrate upon which higher—level languages and protocols can be built. Moreover, these higher—level protocols should be independent of the transport mechanisms (e.g., TCP/IP, email, http, etc.) used.

Environment. The environment in which software agents will be required to work will be distributed, heterogeneous, and dynamic. To provide a communication channel to the outside world in such an environment, a communication language must provide tools for coping with heterogeneity and dynamism. It must support interoperability with other languages and protocols. It must support knowledge discovery in large networks. Finally, it must be easily attachable to legacy systems.

Reliability. A communication language must support reliable and secure communication among agents. Provisions for secure and private exchanges between two agents should be supported. There should be a way to guarantee authentication of agents. Because neither agents nor the underlying transport mechanisms are infallible, a communication language must be robust to inappropriate or malformed messages. The language should support reasonable mechanisms for identifying and signaling errors and warnings.

4.1.3 How KQML Stacks Up

In this section, we evaluate the KQML language as it stands today, relative to our desiderata for agent communication languages.

Form. The only primitives of the language, the performatives, convey the communicative act and the actions to be taken as a result. Thus the form of KQML should be deemed to be declarative. The default format for a KQML message is a linear stream of characters with a Lisp—like syntax. Although this formatting is irrelevant to the function of the language, it makes messages easy to read, easy to parse, and

easy to convert to other formats. The inclusion of named parameter—value pairs has several advantages: optional parameters need not be included; KQML messages are easily converted to an object—oriented or frame—based representation; and extensions using additional parameters are easily added. On the negative side, some potential users find Lisp—like syntax to be undesirable.

Content. The KQML language can be viewed as being divided into three layers: the content layer, the message layer and the communication layer. KQML messages are oblivious to the content they carry. Although in current implementations of the language there is no support for non—ASCII content, there is nothing in the language that would prevent such support. The language offers a minimum set of performatives that covers a basic repertoire of communicative acts. They constitute the message layer of the language and are to be interpreted as speech acts. Although there is no ``right'' necessary and sufficient set of communicative acts, KQML designers tried to find the middle ground between two extremes: 1) providing a small set of primitives thereby requiring overloading at the content level; and 2) providing an extensive set of acts, where inevitably acts will overlap one another and/or embody fine distinctions. The communication layer encodes in the message a set of features that describe lower—level communication parameters (such as the identity of the sender and recipient, and a unique identifier associated with the communication).

Semantics. KQML semantics is still an open issue. For now there are only natural language descriptions of the intended meaning of the performatives and their use (i.e., protocols). An approach that emphasizes the speech act flavor of the communicative acts is a thread of ongoing research~[labrou94a,Smith95,Cohen95.

Labrou and Finin [labrou94a have proposed a specific framework for KQML that defines cognitive states for agents and uses them to describe the performative and associated preconditions, postconditions and satisfiability conditions for felicitous use. In addition, conversation policies are provided in the form of dialogue grammars specifying additional constraints for coherent discourse.

Implementation. There are currently a number of KQML software suites that have been implemented and are in use. These include KATS (Lockheed Martin and UMBC), KAPI (Lockheed, EIT and Stanford), Magenta (Stanford), COOL (University of Toronto), and LogicWare (Crystaliz Inc.). Details on these systems can be found at http://www.cs.umbc.edu/kqml/software/. These implemented systems differ in how they measure up to our implementation desiderata. However, taken as a group, it does appear that implementations are possible that can do well with respect to each of our criteria.

Both the Lockheed KAPI system and the Lockheed Martin/UMBC KATS suite, for example, provide a content—independent message router and a facilitator. Facilitators are specialized KQML agents that maintain information about other agents in their domain, and about those agents' query—answering capabilities (existing versions of facilitators supply only simple registration services). The application must provide a handler function for each performative that is to be processed by the application.

In general, it is not necessary that an application be able to handle all performatives, since not all KQML—speaking applications will be equally powerful. Creating a KQML—speaking interface to an existing application is a matter of providing the appropriate handler functions.

The efficiency of KQML communication has been investigated. Various compression enhancements have been added that cut communication costs by reducing message size, and by eliminating a substantial fraction of symbol lookup and string duplication~[finin94b.

Networking. KQML does address most of the networking desiderata and provides, we believe, a good fit with current networking technology. KQML has been designed to work with multiple transport mechanisms, and implementations have been done that use TCP/IP, SMTP (email), HTTP and CORBA objects to carry messages. KQML agents can be addressed using symbolic names, which are resolved into transport—level addresses by agent name servers. KQML messages can be sent point—to—point; multicasting and broadcasting are possible in any of the transport mechanisms through the use of facilitator class agents. KQML allows both synchronous/asynchronous interactions and blocking/non—blocking message sending on behalf of an application, through assignment of appropriate values for those parameters in a KQML message. We have found the basic primitives to be sufficient to create agents that offer network—oriented services, such as name servers, proxy agents and brokers. Although some work has been done to build higher—level protocols on top of KQML, this remains as a rich area to be explored.

Environment. KQML can use any transport protocol as its transport mechanism (HTTP, SMTP, TCP/IP etc.). Since KQML messages are oblivious to content, there are no restrictions on the content language (beyond the obvious requirement that a handler can be written to process the content of each type of performative). Interoperability with other communication languages remains to be addressed as such languages appear. One such attempt has been made by Davis, whose Agent—K attempts to join KQML with Shoham's Agent Oriented Programming~. The existence of facilitators in the KQML environment can provide the means for knowledge discovery in large networks, especially if facilitators can cooperate with other knowledge discovery applications available in the World Wide Web.

Reliability. Since KQML speaking agents might be imperfect, there are performatives (error and sorry) that can be used to respond to messages that an application cannot process or comprehend; this provides a crude way for an agent to respond to ill—formed, inappropriate or unwanted incoming messages. A more general approach being considered is the development of an ontology of warnings, errors and infelicities that would be appropriate for agents and agent communications languages. Terms in this ontology could be used in the :content field of the rudimentary error and sorry performatives to describe the problem encountered.

The issues of security and authentication are only beginning to be addressed by the KQML community. A security architecture model based on data encryption techniques has been proposed for KQML. In tune with KQML's asynchronous nature, the model expects a secure message to be self authenticating and does not support any challenge/response mechanism to authenticate a message after it has been delivered. The architecture supports two security models, basic and enhanced. The basic security model supports sender authentication, message integrity and data privacy. The enhanced security model additionally supports non—repudiation of origin (proof of sending) and protection from message replay attacks. The enhanced security model also supports frequent change of encryption keys to guard against cipher attacks.

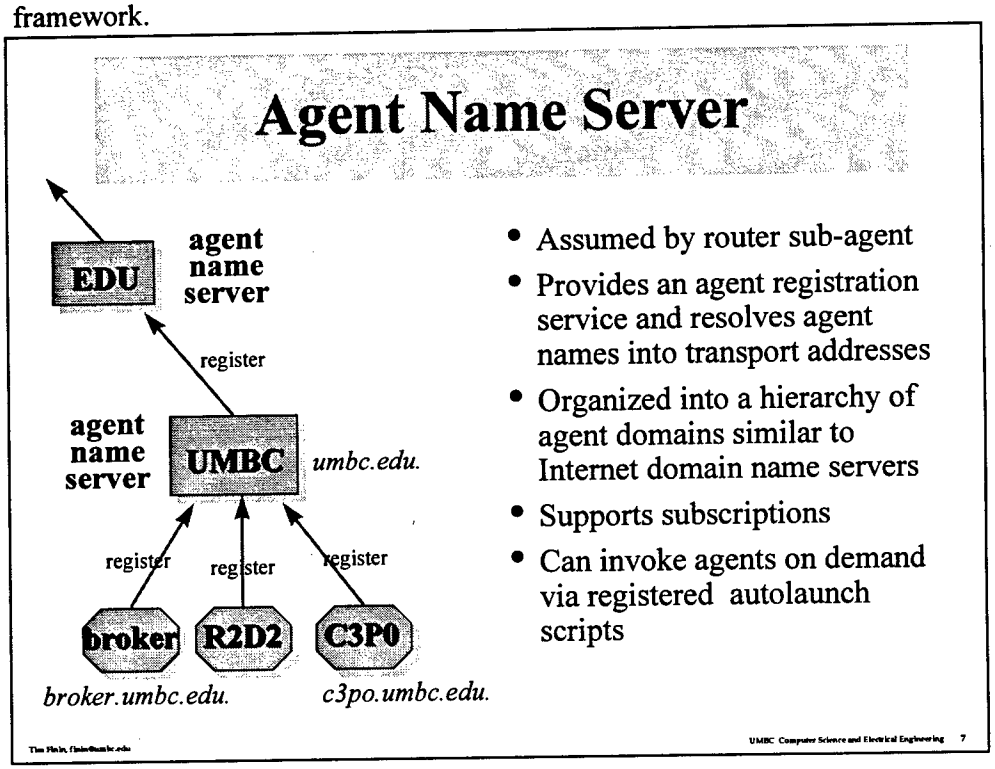
4.1.4 Summary

A good agent communication language has many needs, some of which are in competition. KQML is a new communication language that addresses many (although not all) of these needs. The development of KQML has been marked by an effort to balance the theoretical requirements for a sound and complete framework against the technical demands for efficient, easy—to—use implementations.

The inevitable compromise between the two may not result in a communication language that suits everyone, but we believe that KQML will prove useful in a wide range of intelligent software agent architectures. Additional information about KQML, including papers, language specifications, access to APIs, information on email discussion lists, etc., can be found at http://www.cs.umbc.edu/kqml/.

4.2 Agent Name Services and the Role of the Agent Name Server

We consider the problem of how agents should be named and what kind of software infrastructure is necessary in order to locate an agent given only its name. We assume an agent environment which (1) is dynamic with agents being created and destroyed frequently; (2) undergoes re-organizations with agent groups and sub-groups forming and disbanding; and (3) supports agent communication by any of several transport mechanisms such as tcp/ip, email, http and distributed object systems. This leads us to propose the establishment of *agent domains* which are organized into an *agent domain hierarchy.* Agent name resolution can be done by *agent name servers,* analogous to Internet *domain name servers*. One of the additional benefits from this approach is that it easily supports the definition of *proxy agents*. We sketch how this proposal would impact the KQML agent communication language and protocol and describe an ongoing implementation of a generic *KQML Agent Name Serverr* and its integration into the KATS framework.



Agents need to talk to other agents. If you are an agent A and there is another specific agent B that you want to send a message to, how do you manage it? Well, clearly there is a need for some kind of referential expression that A can use for B and which can be given to the underlying machinery which will convey the message to B. One solution is to use an expression that locates the agent with respect to the message transport system. Examples of such "transport address names" would be a structure which contains an IP address and a port number, or a URL, or an email address for the tcp/ip, http and smtp protocols. This is a common practice in many of our primitive agent systems today. Another approach allows agents to use one of more symbolic names and to provide some kind of mechanism by which names can be registered and associated with their appropriate "transport address name". This approach is only slightly more sophisticated than the first. The name registration can be done in any of several ways, such as hand coding the associations into all of the agents, or broadcasting the

associations over the transport mechanism or assuming the use of "communication facilitator" type agents.
The KQML language and protocol includes special commands (the register and unregister performatives) by which agents can announce the symbolic names by which they wish to be known. Special agents (commonly known as "communication facilitators") traffic in this knowledge and provide a name registration and resolution service. In the Lockheed Martin/UMBC "KQML Agent Technology Software" (KATS) architecture, this name registration and resolution is handled automatically by a generic router sub-agent attached to each agent. From the agents perspective, all it has to do is to specify the set of symbolic names it wished to be known by. The router sub-agent automatically contacts the local "facilitator agent" to register the agent by its symbolic names.
For example, suppose the agent named A wants to send a query to agent B. It passes a KQML form like

(ask-one :from A :to B :content ...)

to its router sub-agent (call it r(A)). This router is responsible, among other things, for resolving the agent name B into an address that can be given to the transport layer for delivery. In KATS, the router checks it's cache to see if it knows how to deliver a message to an agent named "B". If it does, it ships the message out. If not, it sends a KQML query to the local facilitator, asking for the address of an agent named "B". Upon receiving the information, it adds it to its cache and sends off the message.
There are additional wrinkles, of course, such as how to determine when a cache entry is stale and needs to be flushed, but this describes the current arrangement.
Although this approach works quite well as far as it goes, it just does not go very far. The problem is that it only supports communication between two agents if they both register with a common facilitator. There are several possible solutions. All agents could use a single master facilitator, possibly located at a single internet site protected from natural and man-made calamity. Another approach is to have the facilitators share their registration databases. Still a third, and more general, technique involves having the facilitators use a distributed protocol to seek out the contact information on non-local agents. We next describe our protocol for such a distributed agent name resolution scheme.

4.2.1 Distributed agent-name resolution

In our design, we propose to organize agents into "agent domains" in much the same way that the Internet is organized into "host domains". An "agent domain" can be thought of as a collection of agents that are associated with a particular set of facilitator-class agents. In particular, every agent domain must have an "agent domain name server" (or "agent name server" or ANS for short) running. There may be other facilitator-class agents, such as brokers, associated with the agent-domain.
Agent domains will be organized into a hierarchy. Agents will register with an ANS, as shown in figure one. An ANS, being an agent itself, will register with a "parent ANS", resulting in a hierarchy. Each agent will have one or more local names. An agent can also be referred to by its "domain qualified name".

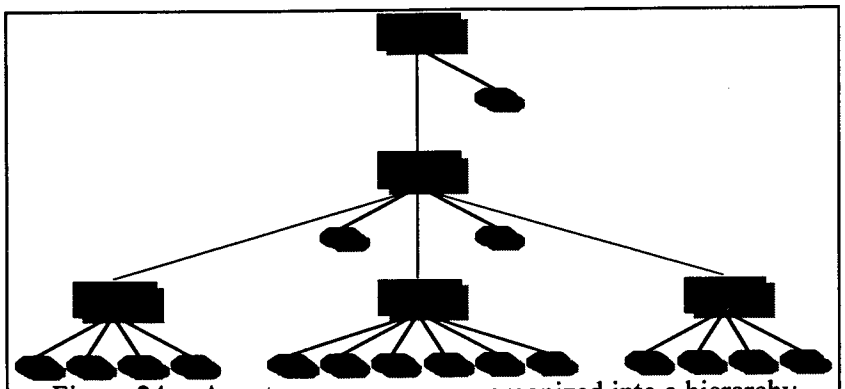


Figure 24 -- Agent name servers are organized into a hierarchy through the registration process

One possibility we might consider is just to "piggy-back" on the existing Internet host structure. For example, why not refer to the agent "colossus" running on the machine "cujo.cs.umbc.edu" as "colossus@cujo.cs.umbc.edu" and assume a standard port for KQML speaking agents. This idea is attractive in that it makes efficient use of a well thought out and implemented architecture. However, there are several problems which argue against this. The primary difficulty is that we do not want to tie KQML and agent communication in general to a single transport mechanism. Current research groups are using a variety of mechanisms to carry KQML messages -- TCP/IP, SMTP, CORBA objects, and HTTP. We would like to continue to keep KQML flexible in this regard. A consequence of this is that we need a general mechanism for naming agents that is independant of the transport mechanism.

4.2.2 What do agent names look like?

We propose a naming scheme similar to the one used for hosts on the Internet. Every agent will have one or more local names optionally followed by a domain qualifier. A local name can be any non-zero length sequence of characters chosen from the character set

{a-z,A-Z,0-1,-,_,.,+,#}

A domain qualifier begins with the character "." and consists of one or more agent domain names separated with a "." character. Thus a fully qualified agent name has the structure:

<local name> . <domain1>.<domain2>.<domain3>...<domainN>

The following would all be valid names for an agent with the local name "colossus" registered in the "umbc.edu." agent domain (and assuming that it is in turn registered in the "edu." domain which is in the top-level "." domain.)[1]

colossus
colossus.umbc
colossus.umbc.edu.

Furthermore, we propose a correspondence between the names of agent domains and agent domain servers. Thus in the above example, agent *colossus* is registered with the ANS with local name *umbc* which is registered with the ANS with local name *edu* which is registered with the global ANS. Thus, the fully qualified name of an agent could be defined by its local name followed by a "." followed by the fully qualified name of its official agent name server.

A number of issues remain to be addressed, these include

- How names are resolved
- Taking names seriously within higher-level abstract protocols such as KQML
- The use of proxy agents to support and implement agent name services.

[1] An alternate is possible, such as one based on URLs. For example, we might choose to represent an agent using "*agent://<domain>.<domain>...<domain>/<local name>* " as in the example "agent://umbc.edu.kqml/colossus".

4.2.3 Name resolution and Proxy Agents

We are exploring what it means to take names seriously -- agents should always communicate with other agents using their names, and not the underlying transport addresses, if known. If you do this, some other things fall out for free, such as proxy agents, i.e., agents which stand in for other agents, for example, such proxy mechanisms have proved exceptionally useful in internet security as well as world wide web operation. The same is true of intelligent agents using KQML.
As an example, suppose we have two agents A and B, both of which use facilitator F. A has proxy agent Pa and B has proxy agent p(b). Suppose A wants to send a message to B. The following events take place:

- a hands off the message to its router r(a).
- r(a) asks f for b's address.
- f gives r(a) the address of p(a), a's proxy.
- r(a) delivers the message to p(a) but the :TO field equals b.
- p(a), knowing that it is a proxy for a (possibly among others) and noticing that it has received a message from a with the :TO field of b, understands that the message is not really intended for it, and asks its router r(p(a)) to deliver it to b.
- r(p(a)) asks f for the b's address.
- F gives r(p(a)) the address of p(p) -- b's proxy).
- r(p(a)) delivers the message to p(b) with the :TO field equals b.
- p(b), knowing that it is a proxy for n (possibly among others) and noticing that the :TO field is b, understands that the message is not really intended for it, and asks its router r(p(b)) to deliver it to b.
- r(p(b)) asks f for the address of b.
- f recognizes that p(b)) is b's proxy so it gives p(b) the real address of b.

This example demonstrates the use of proxy agents for both outgoing messages and messages. The proxy agents may do some additional processing of the messages they get, of course, like logging or traffic analysis, etc. The scenario above is the worst case in that it assumes all of the router caches are empty. Subsequent communications would find the caches filled, so the facilitator would not have to be involved.

Agent A will register itself with the facilitator with a

```
(tell :sender A :receiver facilitator :language kqml
 :content (transport-address :agent A :content (snoopy.cs.umbc.edu 5501)
:language s-expressions :ontology tcp-host-port))
```

If agent A wants to register PA as a proxy agent for itself, it will send a

```
(tell :sender A :receiver facilitator :language kqml
 :content (transport-address :agent PA :content (cs.umbc.edu smtpgw)
 language s-expressions :ontology smtp-domain-user))
```

When the facilitator receives the above message, it should tag this transport address with agent A, rather than agent PA.

4.3 Planning Initiative Information Agent

This section describes the basic feasibility and utility of a Planning Initiaitive Information Agent --- a knowledge server or source capable of handling all requests for information in the transportation planning

domain. We have built a knowledge server prototype which involved the integration of the three knowledge-base/data-base projects: LIM, SIMS, and CoBase, and focused upon the data and information collected for the "Tunisia Scenario". The prototype also tested the robustness of its three component systems in a realistic information environment. It is anticipated that the performance of tasks in the transportation planning domain will require 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 LIM, SIMS, and CoBase systems 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

4.3.1 Technical Approach

LIM, SIMS, and CoBase have been combined in various ways, including both a single Common LISP imageand a client/server architecture in which the LIM server was at a remote site. shared one Loom model of the application domain (the Tunisia Scenario) and the (Oracle) databases. Queries were submitted in the Loom language, extended by the approximation operators proposed by the CoBase project. 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 will be automatically accessed over a network, using the LIM/IDI database interface.

Cooperative query answering was accomplished by CoBase in two ways. First, the user is able to specify explicit relaxation operators in the query. The operators were syntactically transformed to change a cooperative query into a standard LOOM query. Second, if no answer was found, CoBase as directed by the user applied the type abstraction hierarchy and query modification to derive approximate queries that were answered in place of the original query. The modification process consists of query specialization and generalization.

The standard LOOM language was extended to allow explicit cooperative operators and user control over query modification. The explicit operators provided included "approximate", "near-to" and "similar-to". Control operators include "relaxation-order" and "not-relaxable".

The SIMS planner transformed the new semantic query (thus named to stress that it is phrased in high-level domain-model terms, not necessarily referring explicitly to any specific databases or relations) into a series of operations. The operations may consist either of LIM queries directed at individual databases, or of knowledge manipulations performed by the Loom system.

Each LIM query was processed, the relevant database identified and the query translated into an SQL query (or queries) for that database. A graphical user interface was provided to show a trace of the

various stages of a semantic query's relaxation, modification and processing by the system. The interface will support the inspection of the inputs and outputs of each of the components mentioned above. If desired, it will be possible to circumvent the interface entirely, simply handing the system a query and receiving the desired data in return. This is necessary for the case where the system is utilized by a computer program, e.g., a planner. The interface will be developed by the CoBase project team based on the existing SIMS interface, and with help from the SIMS project.

4.3.2 LOOM Interface Module

Intelligent information systems, the integration of artificial intelligence and database technologies, promises to play a significant role in shaping the future of computing. The motivations driving the integration of these two technologies include the need for

access to large amounts of shared data for knowledge processing,
- efficient management of knowledge as well as data, and

intelligent processing of data.

In addition, AI/DB integration at Lockheed Martin Defense Systems is motivated by the desire to preserve the substantial investment in most legacy databases prevalent in the military logistics arena. To that end, a key design criterion was that our integration technology support the use of existing DBMSs as independent system components. The development of large-scale, heterogeneous, distributed intelligent systems for military transportation planning involving a number of discrete cooperating agents has led to a need for persistent storage of knowledge to permit knowledge sharing among those agents and conventional legacy applications. Thus, intelligent systems require access to the actual database structures used regardless of the underlying implementation but translated into structures more amenable to knowledge-based processing. Finally, one key measure of the impact of the overall initiative will be the impact on extending the information structures used for, and retained by, the military logistics planning community resulting in a general need to retain the results of knowledge-based processes for the long-term.

LIM, the LOOM Interface Module[Pastor 1994, Pastor 1992], is being developed in the context of supporting military logistics planning for the United States Transportation Command, which is responsible for planning large-scale movements of troops and materiel. The current planning system is a 70s-vintage batch-oriented system that uses large, heavily-encoded records; the plan for even a moderate-sized operation is enormous, with some containing hundreds of thousands of records, effectively precluding any purely memory-resident storage scheme. The system has recently been extended to use a relational database in place of flat files for some applications, and in some cases, object oriented databases. The designers of the relational DB schema were constrained by a need to adhere fairly closely to the original data format for the plan records, since the planners are quite familiar with the current structure of the data. Our project is thus faced with the task of interfacing a knowledge representation system to a collection of legacy databases that lack a coherent semantic schema. There is also a need to share data and knowledge structures derived by one among a group of intelligent systems with other systems.

LIM uses a view concept model of the application and data domains stated in a high level language (KRSL) and implemented on top of the LOOM knowledge representation system. The view concept model is an extension of the view object model [Wiedehold 1986]. Using LIM, an information agent (see Figure 25) has been built which mediates between knowledge structures defined for use by the intelligent system components being developed and demonstrated as a part of the Planning Initiative. 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 in this article 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 at the point of contact. All three systems can be accessed independently depending on the desired functionality. The Knowledge Query and Manipulation Language (KQML 1994) is used to support this level of communication transparency.

LIM has been used within the Planning Initiative in several experimental prototypes demonstrating intelligent systems technology. The most recent is an experiment conducted over the internet involving agents at Lockheed Martin, USC ISI and UCLA. The demonstration uses information access queries deemed important by end users of current planning system prototypes which, in part, involved answering transportation related questions via a map interface. While LIM has demonstrated considerable functionality required for intelligent access to databases, key challenges of providing efficient access

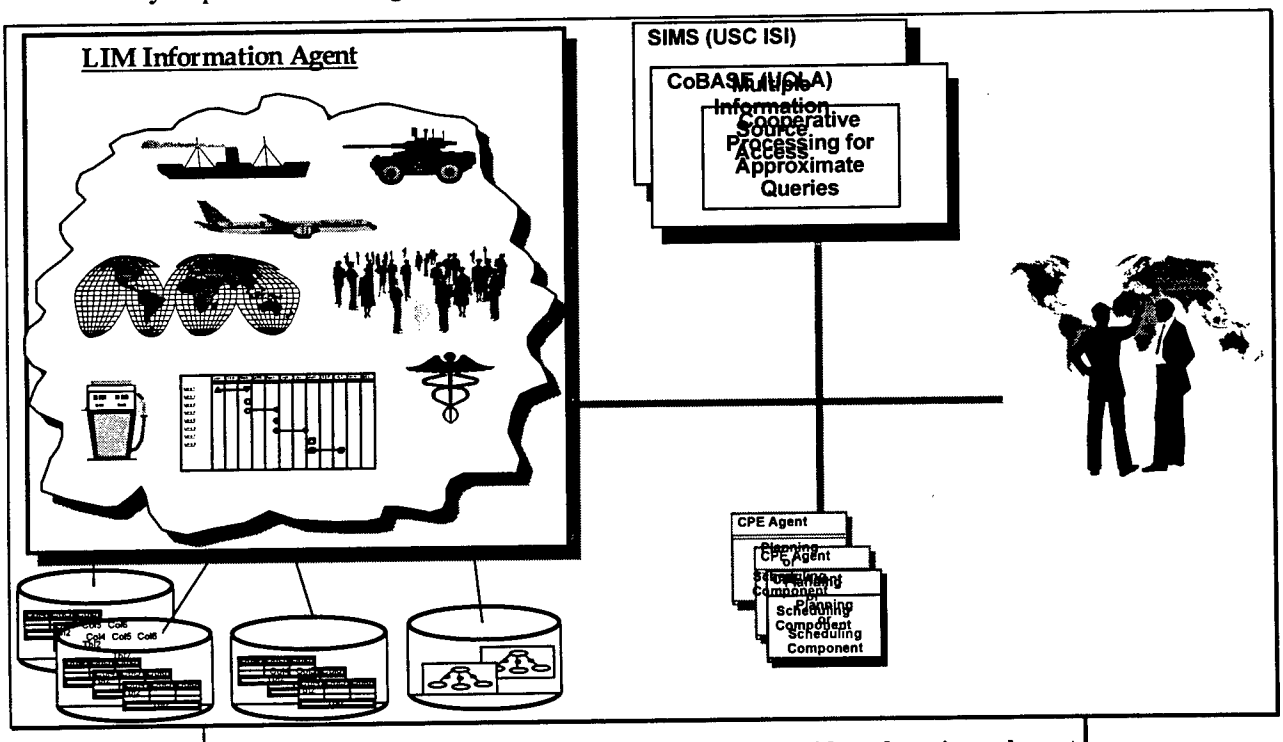


Figure 25. LIM Information Agent. LIM provides domain relevant representations of transportation assets and other resources in a high-level representation. It mediates between the storage structures and the representations used by other intelligent agents.

mediated via large and complex knowledge structures are still of paramount concern. We have developed several versions of the system and have performed extensive performance evaluations and improvements throughout the development of LIM. For example, initial tests conducted using large knowledge structures accessed via LIM required significant processing time which have subsequently have been reduced to nominal execution times. LIM overhead over and above database execution time has been measured at about 40%. In addition, the development and attachment of intelligent system models and advanced representations (e.g., defaults and semantic constraints) required for information modeling are still under active development. The current system is implemented in Common Lisp, however, the methods and techniques are 1) applicable to system development methodologies using commercial tools such as the SNAP development system from Template Software or 2) reimplemented on a ad hoc custom basis in C++.

4.3.3 Cooperative Information Access Research at UCLA

(Note: This section provided by Wes Chu of UCLA)

In conventional databases, if required data is missing, if an exact answer is unavailable, or if a query is not well-formed with respect to the schema, the database just returns a null answer or an error. An intelligent system would be much more resourceful and cooperative, permitting conceptual level queries (containing concepts that may not be expressed in the database schema) when the user does not know the precise schema, providing approximate answers when some data is missing, or even volunteering associative (relevant) information to the query answer.

In ARPI, we have developed Cooperative Query Answering Facility (CoBase) to support such information access. CoBase provides intelligent access to heterogeneous knowledge and data sources. This includes approximate answers when exact answers are not available, summary answers grouped together by category, and answers to high-level conceptual queries [Chu:93,Chu94].

Cooperative answers can be provided by rewriting the query to add variables to the query vector which carry relevant information to the user [Cuppens:88. The rewrite is not a generalization of the query, rather, it extends the query to provide more information. Motro [Motro:88] proposes allowing the user to select the direction of relaxation and thus to indicate which relaxed answers may be of interest. All the above approaches are rule-based, lack of a systematic organization to guide the query transformation process, thus are difficult to scale up.

In ARPI, we have been developing the concept of Type Abstraction Hierarchy (TAH) [Chu94] to provide a structured and scalable approach to query modification and explicit cooperative operators for query relaxation.

TAH provides multilevel object representation which is not captured by the conventional semantic network and object-oriented database approaches for the following reasons: grouping objects into a class and grouping several classes into a super-class only provide a common "title" (type) for the involved objects without concern for the object instance values and without introducing abstract object representations. Grouping several objects together and identifying their aggregation as a single (complex) object does not provide abstract instance representations for its component objects. Therefore, an object-oriented database deals with information only at two general layers: the ***meta-layer*** and the ***instance*** layer. Since forming an object-oriented type hierarchy does not introduce new instance values, it is impossible to introduce an additional instance layer. In the Type Abstraction Hierarchy, instances of a super-type and a sub-type may have different representations and can be viewed at different instance layers. Such multiple layer knowledge representation is essential for cooperative query answering.

Further, we have developed algorithms to automatically generate TAHs based on database instances [Matthew:93,chiang:94], which is important for supporting systems with large databases.

The cooperative operations consist of the following three types: ***context-free***, ***context-sensitive***, and ***control***. For context-free operations, "approximate" operator relaxes the specified values within the approximate range predefined by the user; "between" operator specifies the interval for an attribute, and "within" operator specifies a set membership for an attribute value. For context-sensitive operators, "near-to" is used for specification of geographical nearness; "similar-to" is used to specify a set of objects semantically similar to the given object. It is a multiple attribute operator with a weight assigned to each of the attributes. The attributes may be also be expressed in non-numerical values for specifying the similarity abstractions. For control operators, "relaxation-order" specifies the order of the relaxation; "not-relaxable" specifies the attributes that should not be relaxed; "preference lists" specify the preferred value of the given query conditions; "unacceptable lists" allow users to inform the system not to provide certain answers; and "alternative TAHs" allows users to use the TAHs of their choices.

When an approximate answer is returned, the user may wish for an explanation of how the answer was derived, or may be presented with an annotated relaxation path. A context-based semantic ***nearness*** will be provided, allowing the system to rank the approximate answers (in order of nearness) against the specified query. A user is able to construct his own TAH of any attribute, tuples, or types and is also able to modify the existing TAH in the knowledge base (KB). There is a directory/dictionary of KB that specifies all the available TAHs in the system.

The policy for relaxation control depends on many factors, including user profile, query context, control operators, relaxation level, and the number of answers, as well as rules to combine these factors over multiple attributes.

A Relaxation Manager combines these rules via certain policies (minimizing search time or nearness, for example) to restrict the search for approximate answers.

Often it is desirable to provide additional information relevant to, though not explicitly stated in, a user's query. For example, in finding the location of an airport satisfying the runway length and width specifications, we provide additional information about the runway and weather conditions so that this additional information may help the pilot select a suitable airport to land his aircraft. CoBase employs a case-based approach to build a case library of associative links and their terminations among the attributes of previous queries. When a new query is processed, similar query cases and their associative links are retrieved from the case library and modified to provide additional and relevant information to the new query [Fouque:94]. CoBase stemmed from the Transportation Planning application, which requires querying a large number of databases with different schemas. It has also been implemented on a knowledge-based medical image database, which provides approximate answers to medical queries [Chu:93a].

4.3.4 The SIMS Project at USC/ISI

(Note this section supplied by Yigal Arens and Craig Knoblock of USC ISI)

The Problem

Much productivity is lost today due to the inability of people in business, the military, industry, and academia to identify and access some of the varied stores of information that exist and may be of use in the performance of their tasks. Even when needed information sources *are* found, the effort – and hence cost – that must go into dealing with the different organizations, access and query languages, and different data formats can be enormous. For example, an intelligence analyst may need information about a plane suspected of being involved in drug transport. Information about plane ownership can be found in one database, controlled and maintained by one agency; information about pilot identity and flight plans filed may be found elsewhere; and information about airports and landing strips in areas of interest are located elsewhere yet. Furthermore, the data in different sources may require the use of different keys for identification of the same people or objects and may use different encoding standards. Collecting all the necessary information will require a large amount of effort that would simply be taken away from more productive analysis tasks. Often, the difficulties involved will mean that potentially helpful information will just not be found.

The problem is that locating, retrieving and combining information is an arduous task. Even if one knows where all information that may be relevant to one's task is stored – and that is *not* typically the case – different information sources organize their data differently, require different access and query languages, and use different formats for what is stored in them. The recent government initiative on the National Information Infrastructure will make even greater amounts of information available over electronic networks. The problem of providing access to heterogeneous, distributed information sources must be solved to take advantage of the NII – or even just to make full use of the numerous databases that already exist within even moderately large organizations. Today, people build specialized applications to handle

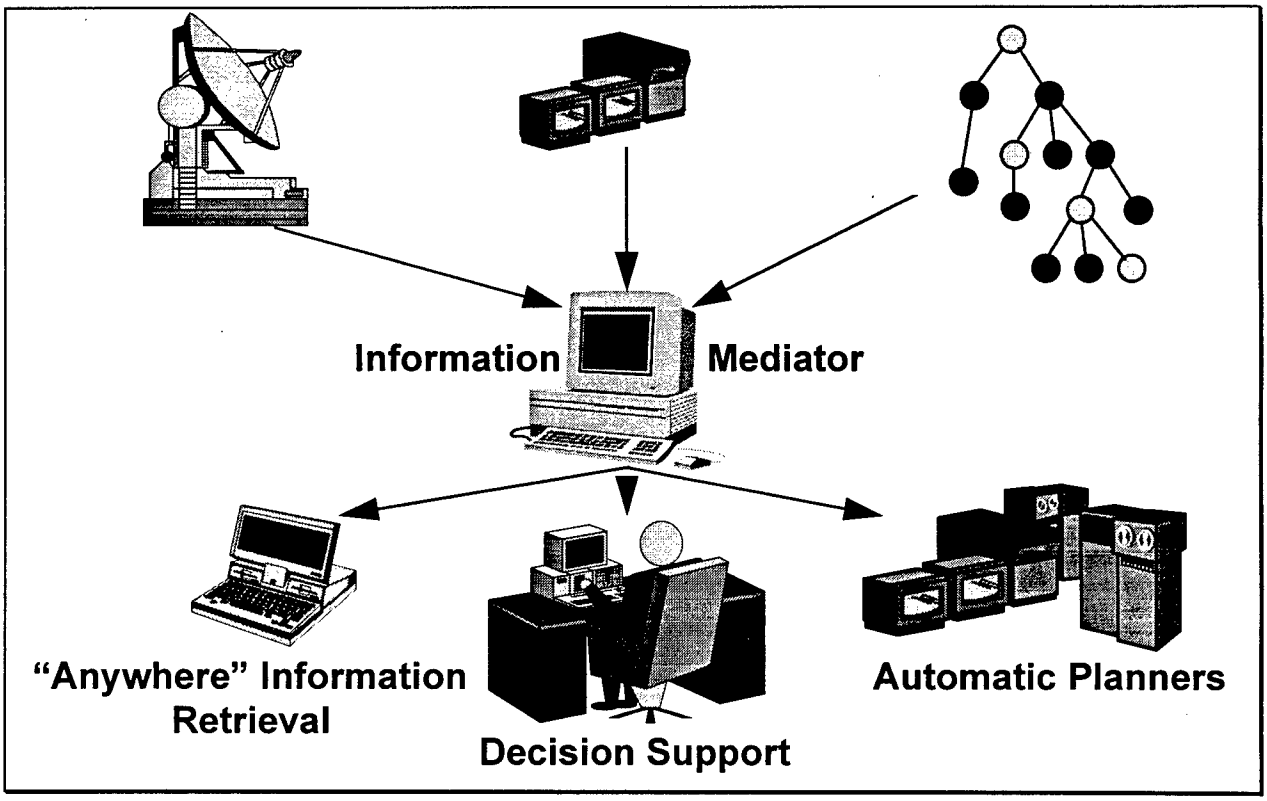


Figure 26. The SIMS architecture supports multiple information sources and requestors for anywhere information retrieval, decision support and automatic planning.

their specific tasks because general tools for integrating and processing information do not exist.

SIMS attempts to solve this problem in its most general form. It is an information mediator that provides an interface between human users or application programs and the information sources to which they need access, locating and integrating information from distributed, heterogeneous sources. SIMS accepts queries in a uniform language, independent of the distribution of information over sources, of the various query languages, the location of information sources, etc. SIMS determines which data sources to use, how to obtain the desired information, how and where to temporarily store and manipulate data, and how to maintain an acceptable level of efficiency in performing its task.

The Approach

In contrast to the standard approach to this problem – building custom systems – SIMS applies and extends a variety of AI techniques and systems to build an intelligent and dynamic interface to the information sources.

SIMS starts out with a model of its domain of expertise, which defines the terminology for querying it, as well as models of all information sources that are available to it. The domain model includes class descriptions, subclass and superclass relationships between classes, roles for each class, and other domain-specific information.

The SIMS project has developed modeling techniques that enable information sources and their characteristics to be described in sufficient detail to permit decisions to be made automatically by the system concerning which information sources should be accessed, when and how. The information source models describe both the contents of the information sources and the relationship between those models and the domain model – i.e., how the classes of objects contained in an information source fit into the general domain model. For each information source the model indicates the data-model used, query language, network location, size estimates, etc., SIMS' models of different information sources are independent, greatly easing the process of extending the system.

Queries to SIMS are also expressed in Loom . They are composed using a vocabulary of general terms in the domain (as defined in the domain model), so there is no need to know or even be aware of the different terms or languages used in the underlying information sources.

The SIMS project has extended AI planning techniques to translate the domain-level query into a series of queries to the underlying information sources. The planning component first selects information sources to be used in answering a query. It then orders sub-queries to the appropriate information sources, selects the location for processing intermediate data, and determines which sub-queries can be executed in parallel.

The SIMS project has extended AI learning research to support a learning component: information about the contents of accessible information sources is collected in the course of executing queries against them. This information is abstracted in the form of semantic rules, which are later used to improve the efficiency of the plans generated to satisfy new user queries.

Change to and addition of information sources is handled by changing or adding models only. The changes will automatically be considered by the planner in producing future plans that utilize information from the modified sources. This greatly facilitates extensibility. SIMS can currently handle Oracle and MUMPS databases, Loom knowledge bases, and certain programs. In principle the SIMS approach supports access to databases, knowledge bases, and even application programs with simple input/output behavior. We are currently extending the variety of information sources supported. All models used by SIMS are written in the Loom language [MacGregor 88, MacGregor 90].

4.3.5 Information Agent Lessons Learned

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 combined system was demonstrated at the San Antonio Planning Initiative Workshop on February 23rd, 1993 and a demonstration using all the components as separate processes on the internet was demonstrated at the Tucson Planning Initiative Workshop in February, 1994.

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 documentation. Further, without the attempt to integrate, some issues would have not been identified as described above in the preceding Sections.

The CoBASE system has evolved to the point where such distributed operation is the normal operational mode. As demonstrated at the 1995 ARPA Software Systems Symposium, the CoBASE system has evolved to where the use of multiple layers of distributed information agents is a profitable manner in which to pursue university research.

The SIMS system has placed some unexpected requirements upon communication API's which are important to note. See also the description of KQML performance issues in Section 3.5 and Section 3.6. SIMS partitions a single query into a set of queries to execute on multiple target information and data resources. The current implementation of SIMS in Common Lisp utilizes a multi-tasking approach, i.e., multiple Common Lisp tasks within the same Common Lisp UNIX process implement parallel problem decomposition, solving and database execution. In part, this means that there are several overlapped executions of LIM to access different databases satisfying a single SIMS query. Significantly, LIM and the underlying Intelligent Database Interface implementations are able to be executed in this multi-threaded manner. In previous configurations of SIMS, LIM was executed in this multi-tasking manner from a single running Common Lisp program. However, the Lockheed Martin KQML implementation had not been developed assuming that an application might be implemented using multi-tasking in Common Lisp. We believe this to be true of other KQML implementations as well.

The Lockheed Martin KQML implementation currently utilizes a connection oriented protocol, i.e., TCP streams. In order to facilitate interaction over the internet and avoid unnecessary and potentially time consuming connect establishment for each KQML message, we introduced a connection cache into our implementation of KQML. When SIMS attempted to execute multiple requests in parallel, given the Lucid Common Lisp implementation of TCP/IP, a single connection attempt to a remote agent could be interrupted by the Common Lisp scheduler and inadvertently hand control to a competing SIMS subtask. The effect is a resource contention over the connection cache leading to indeterminate or errorful results, e.g., connections associated with the wrong information resource. To address this problem, we have developed a version of the Common Lisp KQML implementation which protects this and other critical resources from scheduler interruptions, i.e., they are now identified as critical sections.

5. KAKEE TIE

ISX and Mitre developed a TIE and demonstration for the February, 1994 ARPI Workshop involving a use of the CPE and knowledge server for use by the FORMAT force module composition and editing systems. The FORMAT system makes different performance demands upon the knowledge server, LIM and the databases. FORMAT acts like a batch system in that it loads all force module descriptions from a TPFDD first, in part, to enable display to the end-user. In addition, FORMAT provides additional structural information over and above the information represented in a TPFDD force module representation and must store this information in TPFDD record comment fields. The update process is done for all force modules as a complete rewrite.

The batch-oriented processing is distinctly different than the ad hoc (or real-time) oriented query processing required for other systems such as CoBASE or SIMS. For either of these systems, there are many queries which must be processed, but, the other system pays some attention to the amount of information that is being returned to the user. This is typically constrained via some aspect of the user interface or querying system, aka a limited region query from CoBASE. While we have handled queries within the Infrastructure Prototype effort, e.g., requests for force modules required for support and sustainment planning in the FMERG system, and addressed performance issues, e.g., reduced operating time from several hundreds of seconds for a total run down to about a minute, the additional attributes and different force module structures utilized by FORMAT are about 10x more complex. This 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. The improvement in performance from the original runs of the KAKEE query set can be seen in Chart 2, in which it is also evident that the improvement has been proportional to the size of the result. Improvements in performance over the past year can be seen more clearly in Chart 3, in which the scale has been expanded to make the distinctions among the timings more apparent.

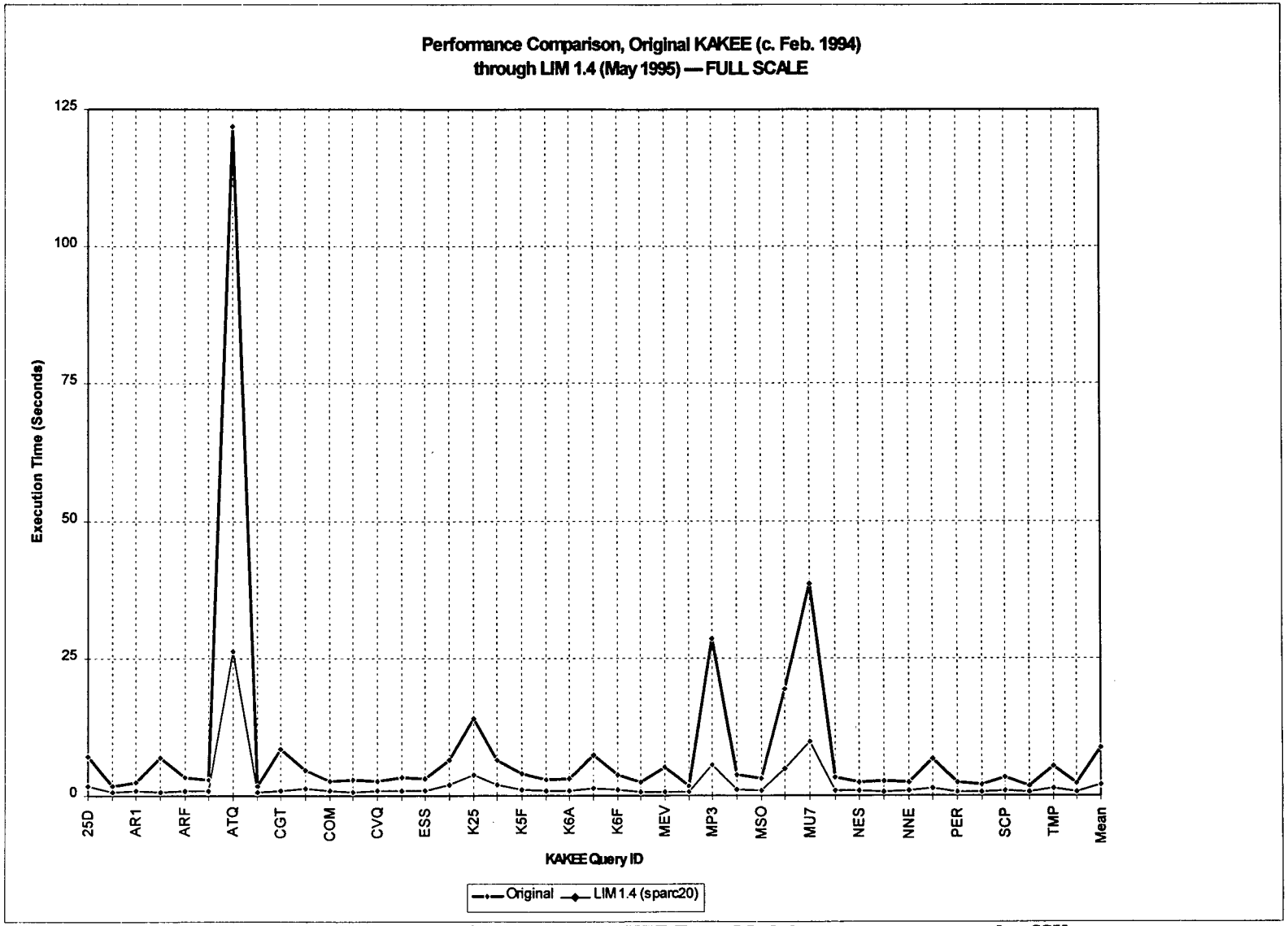


Chart 2. The graph depicts the performance of LIM on the KAKEE Force Module query set, generated at ISX, shortly after its creation using the LIM 1.1 release (February 1994) and using the LIM 1.4 release.

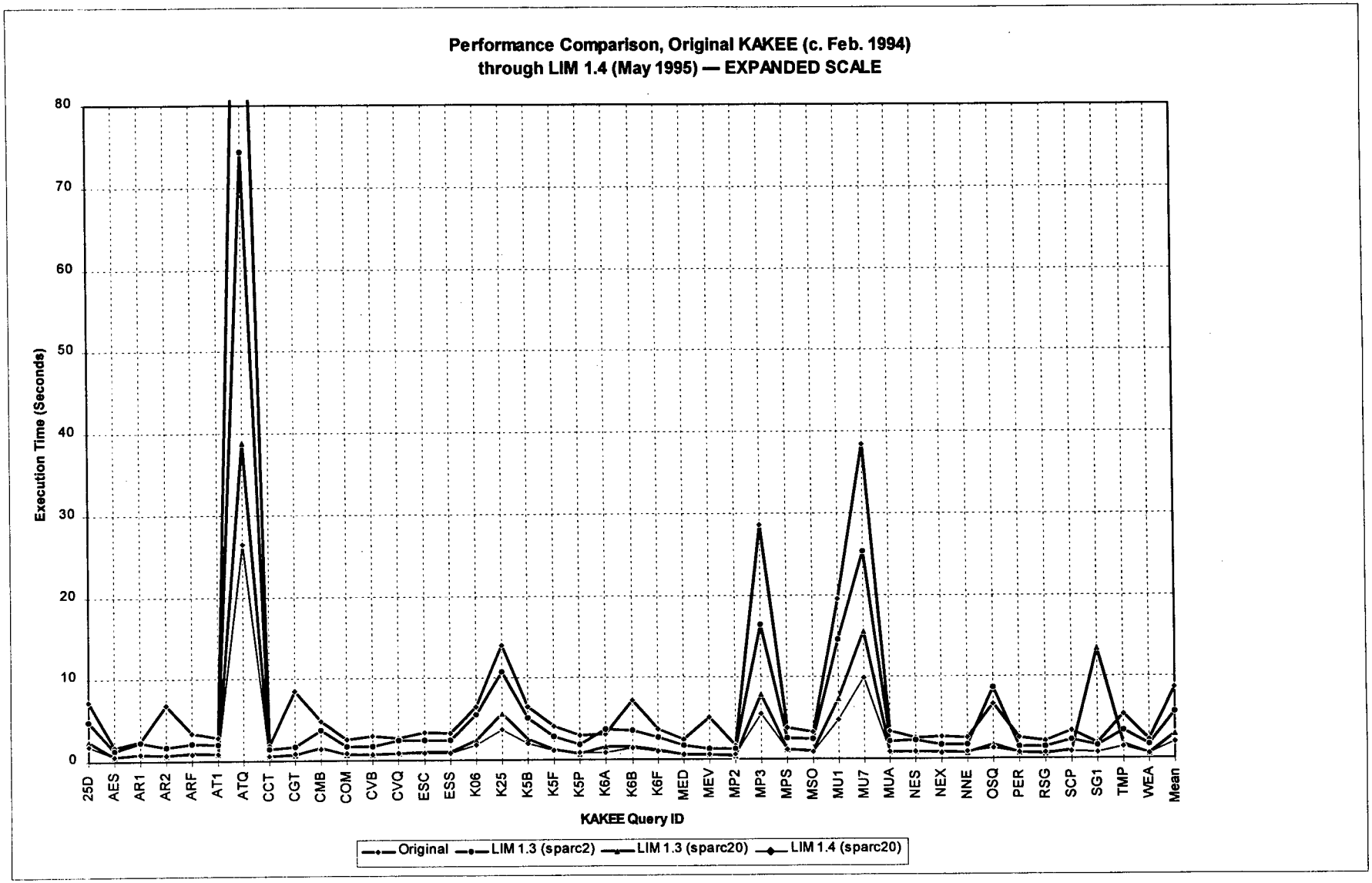


Chart 3. The graph depicts the performance of LIM on the KAKEE Force Module query set using the LIM 1.1 , 1.3, and 1.4 releases, and at two. Note that improvements in performance have been greatest for worst-case queries.

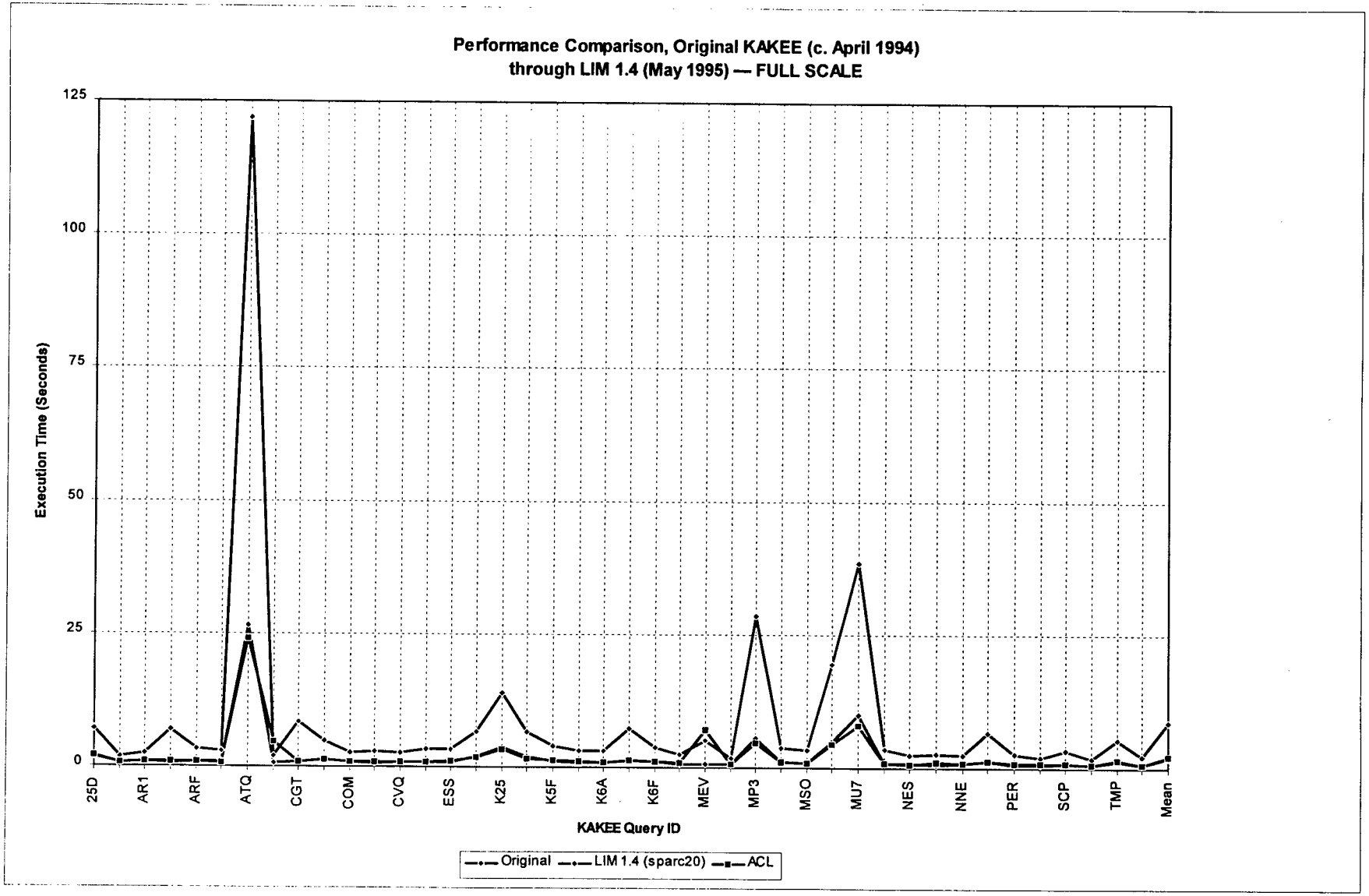


Chart 4The graph depicts the performance of LIM on the KAKEE Force Module query set using the LIM 1.1 , 1.4 releases and the ACL port of LIM 1.4. Note that performance for the ACL port of LIM (without time for performance tuning) is almost identical to that of LIM 1.4 under Lucid Common Lisp.

6. REFERENCES AND BIBLIOGRAPHY

6.1 LIM References

[1] Yigal Arens, "Services and Information Management for Decision Support," *AISIG-90: Proceedings of the Annual AI Systems in Government Conference*, George Washington University, Washington, DC, May, 1990.

[2] Yigal Arens, "Planning and Reformulating Queries for Semantically-Modeled Multidatabase Systems," *Proceedings of the First International Conference on Information and Knowledge Management*, Baltimore, MD, November, 1992.

[3] Thierry Barsalou and Gio Wiederhold, "Applying a Semantic Model to an Immunology Database," 1987.

[4] Thierry Barsalou, "An Object-Based Architecture for Biomedical Expert Database Systems," 1988.

[5] Thierry Barsalou, Arthur M, Keller, Niki Siambela, and Gio Wiederhold, "Updating Relational Databases through Object-Based Views," ACM, 1991.

[6] Ronald Brachman and James Schmolze, "An Overview of the KL/ONE Knowledge Representation System," *Cognitive Science* 9, 1985, pages 171–216.

[7] M. Brodie, J. Mylopoulos, and J. W. Schmidt, editors, *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages*, Springer-Verlag, 1984.

[8] Wesley W. Chu, Andy Y. Hwang, Rei-Chi Lee, Qiming Chen, Matthew Merzbacher, and Herbert Hecht, "Fault Tolerant Distributed Database System via Data Inference," *Proceedings of the Ninth Symposium on Reliable Distributed Systems*, Huntsville, Alabama, October 9–11, 1990.

[9] R. Kowalski, *Logic for Problem Solving*, Elsevier, 1979.

[10] Robert MacGregor and Robert Bates, "The Loom Knowledge Representation Language," *Proceedings of the Knowledge-Based Systems Workshop*, April 1987.

[11] Don McKay, Tim Finin, and Anthony O'Hare, "The Intelligent Database Interface," *Proceedings of the 7th National Conference on Artificial Intelligence*, 1990.

[12] G. Christian Overton, Kimberle Koile, and Jon A. Pastor, "GeneSys: A Knowledge Management System for Molecular Biology," *Computers and DNA*, Santa Fe Institute, G. Bell and T. Marr, editors, Addison-Wesley, Reading, MA, 1990.

[13] Michael Stonebraker and Larry Rowe, *The Postgres Papers*, University of California, Berkeley, 1987.

[14] Gio Wiederhold and R. ElMasri, "The Structural Model for Database Design," in *Entity-Relationship Approach to System Analysis and Design*, pages 237–257, North Holland, 1980.

[15] Gio Wiederhold, "Views, Objects, and Databases," *IEEE Computer*, Vol. 19, no. 12, December 1986, pages 37–44.

6.2 References for KQML

External Interfaces Working Group ARPA Knowledge Sharing Initiative. KQML Overview. Working paper, 1992.

External Interfaces Working Group ARPA Knowledge Sharing Initiative. Specification of the KQML agent-communication language. Working paper, December 1992.

S. Bussmann and J. Mueller. A communication architecture for cooperating agents. Computers and Artificial Intelligence, 12:37--53, 1993.

M. Cutkosky, E. Engelmore, R. Fikes, T. Gruber, M. Genesereth, and W. Mark. PACT: An experiment in integrating concurrent engineering systems. 1992.

Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Trends in cooperative distributed problem solving. IEEE Transactions on Knowledge and Data Engineering, 1(1):63--83, March 1989.

Dan Kuokka et. al. Shade: Technology for knowledge-based collaborative. In AAAI Workshop on AI in Collaborative Design, 1993.

James McGuire et. al. Shade: Technology for knowledge-based collaborative engineering. Journal of Concurrent Engineering: Research and Applications, to appear.

Tim Finin, Rich Fritzson, and Don McKay et. al. An overview of KQML: A knowledge query and manipulation language. Technical report, Department of Computer Science, University of Maryland Baltimore County, 1992.

Tim Finin, Rich Fritzson, and Don McKay. A language and protocol to support intelligent agent interoperability. In Proceedings of the CE& CALS Washington '92 Conference. June 1992.

Tim Finin, Don McKay, Rich Fritzson, and Robin McEntire. KQML: an information and knowledge exchange protocol. In International Conference on Building and Sharing of Very Large-Scale Knowledge Bases, December 1993.

M. Genesereth and R. Fikes et. al. Knowledge interchange format, version 3.0 reference manual. Technical report, Computer Science Department, Stanford University, 1992.

Mike Genesereth. Designworld. In Proceedings of the IEEE Conference on Robotics and Automation, pages 2,785--2,788. IEEE CS Press.

Mike Genesereth. An agent-based approach to software interoperability. Technical Report Logic-91-6, Logic Group, CSD, Stanford University, February 1993.

Carl Hewitt and Jeff Inman. DAI betwixt and between: From "intelligent agents" to open systems science. IEEE Transactions on Systems, Man and Cybernetics, 21(6), December 1991. (Special Issue on Distributed AI).

Michael N. Huhns, David M. Bridgeland, and Natraj V. Arni. A DAI communication aide. Technical Report ACT-RA-317-90, MCC, Austin TX , October 1990.

R. E. Kahn, Digital Library Systems, Proceedings of the Sixth Conference on Artificial Intelligence Applications CAIA-90 (Volume II: Visuals), Santa Barbara CA, pp. 63-64, 1990.

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

Don McKay, Tim Finin, and Anthony O'Hare. The intelligent database interface. In Proceedings of the 7th National Conference on Artificial Intelligence, August 1990.

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

Jeff Y-C Pan and Jay M. Tenenbaum. An intelligent agent framework for enterprise integration. IEEE Transactions on Systems, Man and Cybernetics, 21(6), December 1991. (Special Issue on Distributed AI).

Mike P. Papazoglou and Timos K. Sellis. An organizational framework for cooperating intelligent information systems. International Journal on Intelligent and Cooperative Information Systems, 1(1), (to appear) 1992.

Jon Pastor, Don Mckay and Tim Finin, View-Concepts: Knowledge-Based Access to Databases, First International Conference on Information and Knowledge Management, Baltimore, November 1992.

R. Patil, R. Fikes, P. Patel-Schneider, D. McKay, T. Finin, T. Gruber, and R. Neches. 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 (KR'92), San Mateo, CA, November 1992. Morgan Kaufmann.

J. R. Searle. What is a speech act? In M. Black, editor, From Philosophy in America, pages 221--239. Allen & Unwin, Ort??, 1965.

Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. IEEE Transactions on Computers, C-29(12):1104--1113, December 1980.

Reid G. Smith and Randall Davis. Framework for cooperation in distributed problem solving. IEEE Transactions on System, Man, and Cybernetics, SMC-11(1):61--70, January 1981.

M.Tenenbaum, J. Weber, and T. Gruber. Enterprise integration: Lessons from shade and pact. In C. Petrie, editor, Enterprise Integration Modeling. MIT Press, 1993.

Gio Wiederhold Peter Wegner and Stefano Ceri. Toward megaprogramming. Communications of the ACM, 33(11):89--99, November 1992.

Steven T. C. Wong and John L. Wilson. COSMO: a communication scheme for cooperative knowledge-based systems. IEEE Transactions on Systems, Man and Cybernetics, to appear.

6.3 Information Agent References

LIM References

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

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

[Wiederhold 1986] Gio Wiederhold, "Views, Objects, and Databases," *IEEE Computer*, Vol. 19, no. 12, December 1986, pages 37–44.

CoBASE References

[Chu94] Wsley.~W. Chu and Q.~Chen. A structured approach for cooperative query answering. *IEEE Transactions on Knowledge and Data Engineering*, 6(5):738--749, 1994.

[Chiang:94} Wesley~W. Chu and Kuorong Chiang. Abstraction of high level concepts from numerical values in databases. In *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases*, July 1994.

[Chu:93a] Wesley~W. Chu, A.F. Cardenas, and R.K. Taira. A knowledge-based multimedia medical distributed database system -- {KMeD}. Technical Report 93-0005, UCLA Computer Science Department, 1993.

[Cuppens:88] F.~Cuppens and R.~Demolombe. Cooperative answering: a methodology to provide intelligent access to databases. In *Proceedings of the 2th International Conference on Expert Database Systems*, Virginia, USA, 1988.

[Chu:93] Wesley~W. Chu, M.~A. Merzbacher, and L.~Berkovich. The design and implementation of {CoBase}. In *Proceedings of ACM SIGMOD 93*, pages 517--522, Washington D C., USA, May 1993.

[Chu:94] W.W. Chu, H.~Yang, K.~Chiang, B.~Ribeiro, and G.~Chow. GoGIS: a cooperative geographical information system. In *Proceedings of SPIE Conference on Knowledge-Based Artificial Intelligence Systems in Aerospace and Industry*, Orlando, FL, April 1994.

[Fouque:94] G.~Fouque, W.W. Chu, and H.~Yau. A case-based reasoning approach for associative query answering. In *Proceedings of the Eighth International Symposium on Methodologies for Intelligent Systems*, Charlotte, NC, October 1994.

[Matthew:93] Matthew Merzbacher and Wesley~W. Chu. Pattern-based clustering for database attribute values. In *Proceedings of AAAI Workshop on Knowledge Discovery*, Washington, DC, 1993.

[Mot88] A.~Motro. VAGUE: A user interface to relational databases that permits vague queries. *ACM Transactions on Office Information Systems*, 6(3):187--214, 1988.

SIMS References

1. Arens, Y., Chee, C.Y., Hsu, C-N., and Knoblock, C.A. 1993. Retrieving and Integrating Data from Multiple Information Sources. In International Journal of Intelligent and Cooperative Information Systems. Vol. 2, No. 2. Pp. 127-158.
2. Hsu, C-N., and Knoblock, C.A. 1993. Reformulating Query Plans For Multidatabase Systems. In Proceedings of the Second International Conference of Information and Knowledge Management, Washington, D.C.
3. Arens, Y. and Knoblock, C.A. 1992. Planning and Reformulating Queries for Semantically-Modeled Multidatabase Systems, in Proceedings of the First International Conference on Information and Knowledge Management, Baltimore, MD.
4. Hsu, C-N., and Knoblock, C.A. 1994. Rule Induction for Semantic Query Optimization , in Proceedings of the Eleventh International Conference on Machine Learning, San Mateo, CA.
5. Knoblock, C.A., Arens, Y., and Hsu, C-N. 1994. Cooperating Agents for Information Retrieval, in Proceedings of the Second International Conference on Cooperative Information Systems, Toronto, Ontario, Canada.
6. Knoblock, C.A. 1994. Generating Parallel Execution Plans with a Partial-Order Planner . Artificial Intelligence Planning Systems: Proceedings of the Second International Conference (AIPS94), Chicago, IL.
7. Arens, Y and Knoblock, C.A.. 1994. Intelligent Caching: Selecting, Representing, and Reusing Data in an Information Server. In Proceedings of the International Conference on Information and Knowledge Management (CIKM-94). NIST, Gaithersberg, MD, November 29-December 1, 1994.

Loom References

1. MacGregor, R. A Deductive Pattern Matcher. In Proceedings of AAAI-88, The National Conference on Artificial Intelligence. St. Paul, MN, August 1988.
2. MacGregor, R. The Evolving Technology of Classification-Based Knowledge Representation Systems. In John Sowa (ed.), Principles of Semantic Networks: Explorations in the Representation of Knowledge. Morgan Kaufmann. 1990.

7. PLANNING INITIATIVE REVIEW

Lockheed Martin served on the Planning Initiative working group in preparation for the subsequent Phase III BAA. These were our significant inputs.

7.1 Missed Opportunities

- ***agent-oriented planning and scheduling architectures***

More emphasis needs to be placed on developing planning and scheduling system components in an agent framework (such as that enabled by the ARPA Knowledge Sharing Initiative) so that mixed initiative, incremental, iterative, reactive and dynamic planning and scheduling systems can be developed. This is a key part to enabling the actual development of the visionary architecture and demo. The research being sponsored by ARPA within the KSI and DAI communities should be encouraged to contribute to a Common Integration Environment which subsumes the notions of a Common Prototyping Environment and Common Tools Environment.

An agent environment which equally supports mixed initiative interactions of human and computer-based agents alike is required.

Recommend explicitly soliciting work which builds on other ARPA initiatives and cooperative efforts to build a mixed initiative demo system out of existing PI components. And, encourage the development and running of such a system over the Internet including the existence of firewalll machines, etc.

Note: There has been a fundamental miscommunication about what the prototype developers want and what they get from the PI. The prototype developers want tools or components that can participate in an overall system which is inherently "person in the loop" (aka mixed initiative), "person in control" and reacts well to a constantly changing data environment. The CPE has delivered components which work in a single lock-step flow (a unidirectional pipe). I believe the design and the expense of the communication infrastructure for the CPE encourages this lock-step flow. The message from the scenario dramas and the domain experts, in part because of the use of tidy process models reminiscent of the "waterfall s/w development model", is that a total (i.e. THE TOTAL) system is desired. We missed emphasizing all of the "ooops" and change notice postings in the scenarios which would have led immediately to the mixed initiative problem as being a high priority area two years ago. So, now we have TARGET which really cries out for the incremental tools and the PI that has stand alone systems (for the most part), or, successfully deployed incremental tools which were given poor review when integrated into the CPE because they were unable to be used in the incremental fashion in which they were intended (e.g., TMM).

- ***Lack of a realized Visionary Architecture***

Encourage the development of a running Visionary Demo as the output of a two to three month 'summer workshop' to be hosted at some university site or institute. Participants should include mostly graduate students from the Tier 1 research community who can integrate their systems. The effort should be led by either a visionary end-user with a constrained planning and scheduling problem, or, someone from a government lab. Think of this as a "skunk works" for the PI and run them each summer.

- ***HCI & AMP; task manager***

The mixed initiative system model presumes a human-computer interaction model which has been assumed by prototype developers and CPE developers and Tier 1 researchers. Clearly, for the visionary demo to actually succeed, the right HCI modules and a task manager have to exist. Also, TARGET will eventually fail without a more rationalized HCI manager. Requesting proposals which add one to TARGET

- ***case-based planning***

Encourage the submission of proposals dealing with the capture and elaboration of large case bases. CBR for taking plans off the shelf is hindered by the lack of such qualitatively different case only for the lack of data. The payoff for CBR is so huge in the area. Recommend the use of a CBR solution to 'solving' the PI NEO scenario play.

- ***planning and scheduling with uncertain and missing data***

Planning and scheduling systems have assumed that the data for basic information is 'perfect' (complete consistent, correct, ...) and always accessible. In the real world, such is not the case. Integration with the intelligent database effort would begin to address these issues and provide an interesting experiment to push the planning and scheduling algorithms.

(source: Craig Knoblock - ISI; editor: Don McKay -- Lockheed Martin)

7.2 Structural Model

- ***Make the model effective:***

Establish a "PI Technology Review Board" consisting of the Technology Providers (or representative from each technology area), the Technology Integrator (one representative) and the Demonstration Developer (one or two representatives). The charter of the group is to identify technology transition and insertion opportunities on an ongoing basis as well as to identify to the PI as a whole the requirements for technology transition and insertion. The Technology Integration should be the chair and report to PI program management on progress. Another way to think of this is as a continuing SWAT Team A/B or as a continuing focus group on technology transition and insertion. Within the BAA, if this recommendation were used, include the suggestion to include funding for such participation in PI community activities as options.

Note: While some might see this as the function of the TAB, the TAB is primarily for the ARPA and RL program managers at a program level -- this PITRB is focussed entirely internal to the application of PI technology into the CPE and prototypes developed under the PI and other ARPA programs.

- ***Support the model:***

Foster effective collaborations and cooperation by emphasizing and giving priority to technical integration efforts in the Phase III BAA. These should not be limited to using the current CPE—but should include the elaboration, demonstration and evaluation of a Common Integration Environment. Programmatically, metrics about number of technology components that can be integrated, the cost, time and other intangibles should be readily available. I.e., innovative methods of integration which yield more real integration and direct application of technology.

- ***Keep selling the model:***

Almost everyone's comments have neglected the role of the 'Technology Provider' in their comments, the role of the initial working groups within the PI, or the role of the 'Technology Integrator'. Every research group wants direct contact with end-users, customers, etc as if that gets them access to data, models, scenarios, etc that they can use to demonstrate their research. The charge at the Tucson workshop

to demonstrate relevance has perhaps intensified this. The structural model was initially set up to address the issues and problems the "chaos model" engenders. The Tier 1 areas need to be self-organizing (the role of the "Technology Provider") and to have an effective voice in the planning and execution of the CPE and IFDs (the role of the "Technology Provider" and "Technology Integrator"). Everyone must be reminded of the model and sold it by the program management and see it work effectively. I believe most everyone suggests the "chaos model" because the Structural Model has apparently been abandoned by the Technology Integrators, IFDs and program management.

- ***Keep improving the model:***

Who can deny continuous improvement. PI should clearly develop this as a focus group. But, perhaps the TRP will take care of this....

- ***Use the model:***

Use the SDO and CPR—require that everyone contribute to it and use it. The fact that the IFDs do not use such as a starting point and an integral part of their effort hinders subsequent technology integration.

(source: Yigal Arens - ISI; editor and second: Don McKay --Lockheed Martin)

- ***Technology needs to be a more visible part of the model:***

Technology transfer and insertion should be the primary job of someone in the PI. There is no one who is rewarded for such efforts. In the original statement of the program model, this was the joint responsibility of the demonstration developer, the technology integrator and the technology providers. Transfer into a demonstration effort is difficult and resource intensive, e.g., the demonstration environments select hardware and software platforms which are not available to the rest of the PI because they are selected well after most projects have had to commit, and, commercial software is not available to the research projects.

Recommendation(s): Commit some program funds for supplemental demonstration/transfer efforts which can be applied to fill this gap. Or, encourage the submission of options for technology transfer/insertion in all proposals. And, measure the IFD developer on the number of technology transfer subcontracts they let if they have proposed including such optional efforts.

(source: Wes Chu - UCLA; editor and second: Don McKay --Lockheed Martin)

*U.S. GOVERNMENT PRINTING OFFICE: 1998-610-130-61113

DISTRIBUTION LIST

addresses	number of copies
RAYMOND A LIUZZI RL/C3CA 525 BROOKS RD ROME NY 13441-4505	5
LOCKHEED MARTIN C2 INTEG SYSTEMS ADV SW RESEARCH/ADV INFO SYSTEMS 70 EAST SWEDESFORD RD PAOLI PA 19301	5
ROME LABORATORY/SUL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514	1
ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218	2
ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
RELIABILITY ANALYSIS CENTER 201 MILL ST. ROME NY 13440-8200	1
ROME LABORATORY/C3AB 525 BROOKS RD ROME NY 13441-4505	1
ATTN: RAYMOND TADROS GIDEP P.O. BOX 8000 CORONA CA 91718-8000	1

AFIT ACADEMIC LIBRARY/LDEE
2950 P STREET
AREA B, BLDG 642
WRIGHT-PATTERSON AFB OH 45433-7765 1

ATTN: R.L. DENISON
WRIGHT LABORATORY/MLPO, BLDG. 651
3005 P STREET, STE 6
WRIGHT-PATTERSON AFB OH 45433-7707 1

ATTN: GILBERT G. KUPERMAN
AL/CFHI, BLDG. 248
2255 H STREET
WRIGHT-PATTERSON AFB OH 45433-7022 1

OL AL HSC/HRG, BLDG. 190
2698 G STREET
WRIGHT-PATTERSON AFB OH 45433-7604 1

AUL/LSAD
600 CHENNAULT CIRCLE, BLDG. 1405
MAXWELL AFB AL 36112-6424 1

US ARMY STRATEGIC DEFENSE COMMAND
CSSD-IM-PA
P.O. BOX 1500
HUNTSVILLE AL 35807-3801 1

COMMANDING OFFICER
NCCOSC RDT&E DIVISION
ATTN: TECHNICAL LIBRARY,CODE 0274
53560 HULL STREET
SAN DIEGO CA 92152-5001 1

COMMANDER, TECHNICAL LIBRARY
4747000/C0223
NAVAIRWARCENWPNDIV
1 ADMINISTRATION CIRCLE
CHINA LAKE CA 93555-6001

1

SPACE & NAVAL WARFARE SYSTEMS
COMMAND (PMW 178-1)
2451 CRYSTAL DRIVE
ARLINGTON VA 22245-5200

2

SPACE & NAVAL WARFARE SYSTEMS
COMMAND, EXECUTIVE DIRECTOR (PD13A)
ATTN: MR. CARL ANDRIANI
2451 CRYSTAL DRIVE
ARLINGTON VA 22245-5200

1

SPACE & NAVAL WARFARE SYSTEMS CMD
ATTN: PMW163-1 (R. SKIANO) OT-1 ROOM 1044A
53560 HULL ST
SAN DIEGO, CA 92152-5002

1

CDR, US ARMY MISSILE COMMAND
RSIC, BLDG. 4484
AMSMI-RD-CS-R, DOCS
REDSTONE ARSENAL AL 35898-5241

2

ADVISORY GROUP ON ELECTRON DEVICES
SUITE 500
1745 JEFFERSON DAVIS HIGHWAY
ARLINGTON VA 22202

1

REPORT COLLECTION, CIC-14
MS P364
LOS ALAMOS NATIONAL LABORATORY
LOS ALAMOS NM 87545

1

AEDC LIBRARY
TECHNICAL REPORTS FILE
100 KINDEL DRIVE, SUITE C211
ARNOLD AFB TN 37389-3211

1

COMMANDER
USAISC
ASHC-IMD-L, BLDG 61801
FT HUACHUCA AZ 85613-5000

1

US DEPT OF TRANSPORTATION LIBRARY
FB10A, M-457, RM 930
800 INDEPENDENCE AVE, SW
WASH DC 22591 — 1

AIR WEATHER SERVICE TECHNICAL
LIBRARY (FL 4414)
859 BUCHANAN STREET
SCOTT AFB IL 62225-5118 — 1

AFIWC/MSO
102 HALL BLVD, STE 315
SAN ANTONIO TX 78243-7016 — 1

SOFTWARE ENGINEERING INSTITUTE
CARNEGIE MELLON UNIVERSITY
4500 FIFTH AVENUE
PITTSBURGH PA 15213 — 1

NSA/CSS
K1
FT MEADE MD 20755-6000 — 1

DCMAD/WICHITA/GKEP
SUITE B-34
401 N MARKET STREET
WICHITA KS 67202-2095 — 1

PHILLIPS LABORATORY
PL/TL (LIBRARY)
5 WRIGHT STREET
HANSCOM AFB MA 01731-3004 — 1

THE MITRE CORPORATION
ATTN: E. LADURE
D460
202 BURLINGTON RD
BEDFORD MA 01732 — 1

OUSD(P)/DTSA/DUTD
ATTN: PATRICK G. SULLIVAN, JR.
400 ARMY NAVY DRIVE
SUITE 300
ARLINGTON VA 22202 — 2

SOFTWARE ENGR'G INST TECH LIBRARY 1
ATTN: MR DENNIS SMITH
CARNEGIE MELLON UNIVERSITY
PITTSBURGH PA 15213-3890

USC-ISI 1
ATTN: DR ROBERT M. BALZER
4676 ADMIRALTY WAY
MARINA DEL REY CA 90292-6695

KESTREL INSTITUTE 1
ATTN: DR CORDELL GREEN
1801 PAGE MILL ROAD
PALO ALTO CA 94304

ROCHESTER INSTITUTE OF TECHNOLOGY 1
ATTN: PROF J. A. LASKY
1 LOMB MEMORIAL DRIVE
P.O. BOX 9887
ROCHESTER NY 14613-5700

WESTINGHOUSE ELECTRONICS CORP 1
ATTN: MR DENNIS BIELAK
ELECTRONICS SYSTEMS GROUP
P.O. BOX 746, MAIL STOP 432
BALTIMORE MD 21203

AFIT/ENG 1
ATTN:TOM HARTRUM
WPAFB OH 45433-6583

THE MITRE CORPORATION 1
ATTN: MR EDWARD H. BENSLEY
BURLINGTON RD/MAIL STOP A350
BEDFORD MA 01730

UNIV OF ILLINOIS, URBANA-CHAMPAIGN 1
ATTN: DR MEHDI HARANDI
DEPT OF COMPUTER SCIENCES
1304 W. SPRINGFIELD/240 DIGITAL LAB
URBANA IL 61801

HONEYWELL, INC. 1
ATTN: MR BERT HARRIS
FEDERAL SYSTEMS
7900 WESTPARK DRIVE
MCLEAN VA 22102

SOFTWARE ENGINEERING INSTITUTE 1
ATTN: MR WILLIAM E. HEFLEY
CARNEGIE-MELLON UNIVERSITY
SEI 2218
PITTSBURGH PA 15213-38990

UNIVERSITY OF SOUTHERN CALIFORNIA 1
ATTN: DR W. LEWIS JOHNSON
INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY/SUITE 1001
MARINA DEL REY CA 90292-6695

COLUMBIA UNIV/DEPT COMPUTER SCIENCE 1
ATTN: DR GAIL E. KAISER
450 COMPUTER SCIENCE BLDG
500 WEST 120TH STREET
NEW YORK NY 10027

SOFTWARE PRODUCTIVITY CONSORTIUM 1
ATTN: MR ROBERT LAI
2214 ROCK HILL ROAD
HERNDON VA 22070

AFIT/ENG 1
ATTN: DR GARY B. LAMONT
SCHOOL OF ENGINEERING
DEPT ELECTRICAL & COMPUTER ENGRG
WPAFB OH 45433-6583

NSA/OFC OF RESEARCH 1
ATTN: MS MARY ANNE OVERMAN
9800 SAVAGE ROAD
FT GEORGE G. MEADE MD 20755-6000

AT&T BELL LABORATORIES 1
ATTN: MR PETER G. SELFRIDGE
ROOM 3C-441
600 MOUNTAIN AVE
MURRAY HILL NJ 07974

VITRO CORPORATION 1
ATTN: MR ROBERT A. SMALL
14000 GEORGIA AVENUE
SILVER SPRING MD 20906-2972

ODYSSEY RESEARCH ASSOCIATES, INC. 1
ATTN: MS MAUREEN STILLMAN
301A HARRIS B. DATES DRIVE
ITHACA NY 14850-1313

WRDC/AAAF-3 1
ATTN: JAMES P. WEBER, CAPT, USAF
AERONAUTICAL SYSTEMS CENTER
WPAFB OH 45433-6543

TEXAS INSTRUMENTS INCORPORATED 1
ATTN: DR DAVID L. WELLS
P.O. BOX 655474, MS 238
DALLAS TX 75265

BOEING COMPUTER SERVICES 1
ATTN: DR PHIL NEWCOMB
MS 7L-64
P.O. BOX 24346
SEATTLE WA 98124-0346

LOCKHEED SOFTWARE TEHNOLOGY CENTER 1
ATTN: MR HENSON GRAVES
ORG. 96-10 BLDG 254E
3251 HANOVER STREET
PALO ALTO CA 94304-1191

TEXAS A & M UNIVERSITY 1
ATTN: DR PAULA MAYER
KNOWLEDGE BASED SYSTEMS LABORATORY
DEPT OF INDUSTRIAL ENGINEERING
COLLEGE STATION TX 77843

KESTREL DEVELOPMENT CORPORATION 1
ATTN: DR RICHARD JULLIG
3260 HILLVIEW AVENUE
PALO ALTO CA 94304

ARPA/SISTO 1
ATTN: DR KIRSTIE BELLMAN
3701 N FAIRFAX DRIVE
ARLINGTON VA 22203-1714

LOCKHEED D/96-10 B/254E 1
ATTN: JACKY COMBS
3251 HANOVER STREET
PALO ALTO CA 94304-1191

NASA/JOHNSON SPACE CENTER 1
ATTN: CHRIS CULBERT
MAIL CODE PT4
HOUSTON TX 77058

SAIC 1
ATTN: LANCE MILLER
MS T1-6-3
PO BOX 1303 (OR 1710 GOODRIDGE DR)
MCLEAN VA 22102

STERLING IMD INC. 1
KSC OPERATIONS
ATTN: MARK MAGINN
BEECHES TECHNICAL CAMPUS/RT 26 N.
ROME NY 13440

NAVAL POSTGRADUATE SCHOOL 1
ATTN: BALA RAMESH
CODE AS/RS
ADMINISTRATIVE SCIENCES DEPT
MONTEREY CA 93943

HUGHES AIRCRAFT COMPANY 1
ATTN: GERRY BARKSDALE
P. O. BOX 3310
BLDG 618 MS E215
FULLERTON CA 92634

SCHLUMBERGER LABORATORY FOR 1
COMPUTER SCIENCE
ATTN: DR. GUILLERMO ARANGO
8311 NORTH FM620
AUSTIN, TX 78720

MOTOROLA, INC. 1
ATTN: MR. ARNOLD PITTLER
3701 ALGONQUIN ROAD, SUTE 601
ROLLING MEADOWS, IL 60008

DECISION SYSTEMS DEPARTMENT 1
ATTN: PROF WALT SCACCHI
SCHOOL OF BUSINESS
UNIVERSITY OF SOUTHERN CALIFORNIA
LOS ANGELES, CA 90089-1421

SOUTHWEST RESEARCH INSTITUTE 1
ATTN: BRUCE REYNOLDS
6220 CULEBRA ROAD
SAN ANTONIO, TX 78228-0510

NATIONAL INSTITUTE OF STANDARDS 1
AND TECHNOLOGY
ATTN: CHRIS DABROWSKI
ROOM A266, BLDG 225
GAITHSBURG MD 20899

EXPERT SYSTEMS LABORATORY
ATTN: STEVEN H. SCHWARTZ
NYNEX SCIENCE & TECHNOLOGY
500 WESTCHESTER AVENUE
WHITE PLANS NY 20604

1

NAVAL TRAINING SYSTEMS CENTER
ATTN: ROBERT BREAUX/CODE 252
12350 RESEARCH PARKWAY
ORLANDO FL 32826-3224

1

CENTER FOR EXCELLENCE IN COMPUTER-
AIDED SYSTEMS ENGINEERING
ATTN: PERRY ALEXANDER
2291 IRVING HILL ROAD
LAWRENCE KS 66049

1

SOFTWARE TECHNOLOGY SUPPORT CENTER
ATTN: MAJ ALAN K. MILLER
OGDEN ALC/TISE
BLDG 100, BAY G
HILL AFB, UTAH 84056

1

DR JAMES ALLEN
COMPUTER SCIENCE DEPT/BLDG RM 732
UNIV OF ROCHESTER
WILSON BLVD
ROCHESTER NY 14627

1

DR YIGAL ARENS
USC-ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

1

DR RAY BAREISS
THE INST. FOR LEARNING SCIENCES
NORTHWESTERN UNIV
1890 MAPLE AVE
EVANSTON IL 60201

1

DR MARIE A. BIENKOWSKI
SRI INTERNATIONAL
333 RAVENSWOOD AVE/EK 337
MENLO PRK CA 94025

1

DR PIERO P. BONISSONE
GE CORPORATE RESEARCH & DEVELOPMENT
BLDG K1-RM 5C-32A
P. O. BOX 8
SCHENECTADY NY 12301

MR. DAVID BROWN 1
MITRE
EAGLE CENTER 3, SUITE 8
O'FALLON IL 62269

DR MARK BURSTEIN 1
BBN SYSTEMS & TECHNOLOGIES
10 MOULTON STREET
CAMBRIDGE MA 02138

DR GREGG COLLINS 1
INST FOR LEARNING SCIENCES
1890 MAPLE AVE
EVANSTON IL 60201

DR STEPHEN E. CROSS 1
SCHOOL OF COMPUTER SCIENCE
CARNEGIE MELLON UNIVERSITY
PITTSBURGH PA 15213

DR THOMAS CHEATHAM 1
HARVARD UNIVERSITY
DIV OF APPLIED SCIENCE
AIKEN, RM 104
CAMBRIDGE MA 02138

MS. LAURA DAVIS 1
CODE 5510
NAVY CTR FOR APPLIED RES IN AI
NAVAL RESEARCH LABORATORY
WASH DC 20375-5337

DR THOMAS L. DEAN 1
BROWN UNIVERSITY
DEPT OF COMPUTER SCIENCE
P.O. BOX 1910
PROVIDENCE RI 02912

DR WESLEY CHU 1
COMPUTER SCIENCE DEPT
UNIV OF CALIFORNIA
LOS ANGELES CA 90024

DR PAUL R. COHEN 1
UNIV OF MASSACHUSETTS
COINS DEPT
LEDERLE GRC
AMHERST MA 01003

DR JON DOYLE
LABORATORY FOR COMPUTER SCIENCE
MASS INSTITUTE OF TECHNOLOGY
545 TECHNOLOGY SQUARE
CAMBRIDGE MA 02139 1

DR. BRIAN DRABBLE
AI APPLICATIONS INSTITUTE
UNIV OF EDINBURGH/80 S. BRIDGE
EDINBURGH EH1 LHN
UNITED KINGDOM 1

MR. SCOTT FOUSE
ISX CORPORATION
4353 PARK TERRACE DRIVE
WESTLAKE VILLAGE CA 91361 1

MR. STU DRAPER
MITRE
EAGLE CENTER 3, SUITE 8
O'FALLON IL 62269 1

DR MARK FOX
DEPT OF INDUSTRIAL ENG
UNIV OF TORONTO
4 TADDLE CREEK ROAD
TORONTO, ONTARIO, CANADA 1

MR. GARY EDWARDS
ISX CORPORATION
2000 N 15TH ST, SUITE 1000
ARLINGTON, VA 22201 1

MR. RUSS FREW
GENERAL ELECTRIC
MOORESTOWN CORPORATE CENTER
BLDG ATK 145-2
MOORESTOWN NJ 08057 1

DR MICHAEL FEHLING
STANFORD UNIVERSITY
ENGINEERING ECO SYSTEMS
STANFORD CA 94305 1

RICK HAYES-ROTH
CIMFLEX-TEKNOWLEDGE
1810 EMBARCADERO RD
PALO ALTO CA 94303 1

DR JIM HENDLER
UNIV OF MARYLAND
DEPT OF COMPUTER SCIENCE
COLLEGE PARK MD 20742 — 1

MS. YOLANDA GIL
USC/ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292 — 1

MR. MORTON A. HIRSCHBERG, DIRECTOR
US ARMY RESEARCH LABORATORY
ATTN: AMSRL-CI-CB
ABERDEEN PROVING GROUND MD
21005-5066 — 1

MR. MARK A. HOFFMAN
ISX CORPORATION
1165 NORTHCHASE PARKWAY
MARIETTA GA 30067 — 1

DR RON LARSEN
NAVAL CMD, CONTROL & OCEAN SUR CTR
RESEARCH, DEVELOP, TEST & EVAL DIV
CODE 444
SAN DIEGO CA 92152-5000 — 1

DR CRAIG KNOBLOCK
USC-ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292 — 1

MR. RICHARD LOWE (AP-10)
SRA CORPORATION
2000 15TH STREET NORTH
ARLINGTON VA 22201 — 1

MR. TED C. KRAL
BBN SYSTEMS & TECHNOLOGIES
4015 HANCOCK STREET, SUITE 101
SAN DIEGO CA 92110 — 1

DR JOHN LOWRENCE
SRI INTERNATIONAL
ARTIFICIAL INTELLIGENCE CENTER
333 RAVENSWOOD AVE
MENLO PARK CA 94025 — 1

DR. ALAN MEYROWITZ 1
NAVAL RESEARCH LABORATORY/CODE 5510
4555 OVERLOOK AVE
WASH DC 20375

ALICE MULVEHILL 1
BBN
10 MOULTON STREET
CAMBRIDGE MA 02238

DR ROBERT MACGREGOR 1
USC/ISI
4676 ADMIRALTY WAY
MARINA DEL REY CA 90292

DR DREW MCDERMOTT 1
YALE COMPUTER SCIENCE DEPT
P.O. BOX 2158, YALE STATION
51 PROPSPECT STREET
MEW HAVEN CT 06520

DR DOUGLAS SMITH 1
KESTREL INSTITUTE
3260 HILLVIEW AVE
PALO ALTO CA 94304

DR. AUSTIN TATE 1
AI APPLICATIONS INSTITUTE
UNIV OF EDINBURGH
80 SOUTH BRIDGE
EDINBURGH EH1 1HN - SCOTLAND

DIRECTOR 1
DARPA/ITO
3701 N. FAIRFAX DR., 7TH FL
ARLINGTON VA 22209-1714

DR STEPHEN F. SMITH 1
ROBOTICS INSTITUTE/CMU
SCHENLEY PRK
PITTSBURGH PA 15213

DR. ABRAHAM WAKSMAN 1
AFOSR/NM
110 DUNCAN AVE., SUITE B115
BOLLING AFB DC 20331-0001

DR JONATHAN P. STILLMAN 1
GENERAL ELECTRIC CRD
1 RIVER RD, RM K1-5C31A
P. O. BOX 8
SCHENECTADY NY 12345

DR EDWARD C.T. WALKER 1
BBN SYSTEMS & TECHNOLOGIES
10 MOULTON STREET
CAMBRIDGE MA 02138

DR BILL SWARTOUT 1
USC/ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

GIO WIEDERHOLD 1
STANFORD UNIVERSITY
DEPT OF COMPUTER SCIENCE
438 MARGARET JACKS HALL
STANFORD CA 94305-2140

DR KATIA SYCARA/THE ROBOTICS INST 1
SCHOOL OF COMPUTER SCIENCE
CARNEGIE MELLON UNIV
DOHERTY HALL RM 3325
PITTSBURGH PA 15213

DR DAVID E. WILKINS 1
SRI INTERNATIONAL
ARTIFICIAL INTELLIGENCE CENTER
333 RAVENSWOOD AVE
MENLO PARK CA 94025

DR. PATRICK WINSTON 1
MASS INSTITUTE OF TECHNOLOGY
RM NE43-817
545 TECHNOLOGY SQUARE
CAMBRIDGE MA 02139

DR JOHN P. SCHILL 1
ARPA/ISO
3701 N FAIRFAX DRIVE
ARLINGTON VA 22203-1714

DR STEVE ROTH 1
CENTER FOR INTEGRATED MANUFACTURING
THE ROBOTICS INSTITUTE
CARNEGIE MELLON UNIV
PITTSBURGH PA 15213-3890

DR YOAV SHOHAM
STANFORD UNIVERSITY
COMPUTER SCIENCE DEPT
STANFORD CA 94305
1

MR. MIKE ROUSE
AFSC
7800 HAMPTON RD
NORFOLK VA 23511-6097
1

DR LARRY BIRNBAUM
NORTHWESTERN UNIVERSITY
ILS
1890 MAPLE AVE
EVANSTON IL 60201
1

MR. LEE ERMAN
CIMFLEX TECKNOWLEDGE
1810 EMBARCARDERO RD
PALO ALTO CA 94303
1

DR MATTHEW L. GINSBERG
CIRL, 1269
UNIVERSITY OF OREGON
EUGENE OR 97403
5

MR IRA GOLDSTEIN
OPEN SW FOUNDATION RESEARCH INST
ONE CAMBRIDGE CENTER
CAMBRIDGE MA 02142
1

MR JEFF GROSSMAN, CO
NCCOSC RDTE DIV 44
5370 SILVERGATE AVE, ROOM 1405
SAN DIEGO CA 92152-5146
1

JAN GUNTHER
ASCENT TECHNOLOGY, INC.
64 SIDNEY ST, SUITE 380
CAMBRIDGE MA 02139
1

DR LYNETTE HIRSCHMAN
MITRE CORPORATION
202 BURLINGTON RD
BEDFORD MA 01730
1

DR ADELE E. HOWE
COMPUTER SCIENCE DEPT
COLORADO STATE UNIVERSITY
FORT COLLINS CO 80523 — 1

DR LESLIE PACK KAELBLING
COMPUTER SCIENCE DEPT
BROWN UNIVERSITY
PROVIDENCE RI 02912 — 1

DR SUBBARAO KAMBHAMPATI
DEPT OF COMPUTER SCIENCE
ARIZONA STATE UNIVERSITY
TEMPE AZ 85287-5406 — 1

MR THOMAS E. KAZMIERCZAK
SRA CORPORATION
331 SALEM PLACE, SUITE 200
FAIRVIEW HEIGHTS IL 62208 — 1

DR CARLA GOMES
ROME LABORATORY/C3CA
525 BROOKS RD
ROME NY 13441-4505 — 1

DR MARK T. MAYBURY
ASSOCIATE DIRECTOR OF AI CENTER
ADVANCED INFO SYSTEMS TECH G041
MITRE CORP, BURLINGTON RD, MS K-329
BEDFORD MA 01730 — 1

MR DONALD P. MCKAY
PARAMAX/UNISYS
P O BOX 517
PAOLI PA 19301 — 1

DR KAREN MYERS
AI CENTER
SRI INTERNTIONAL
333 RAVENSWOOD
MENLO PARK CA 94025 — 1

DR MARTHA E POLLACK
DEPT OF COMPUTER SCIENCE
UNIVERSITY OF PITTSBURGH
PITTSBURGH PA 15260 — 1

DR RAJ REDDY
SCHOOL OF COMPUTER SCIENCE
CARNEGIE MELLON UNIVERSITY
PITTSBURGH PA 15213 — 1

DR EDWINA RISSLAND
DEPT OF COMPUTER & INFO SCIENCE
UNIVERSITY OF MASSACHUSETTS
AMHERST MA 01003 — 1

DR MANUELA VELOSO
CARNEGIE MELLON UNIVERSITY
SCHOOL OF COMPUTER SCIENCE
PITTSBURGH PA 15213-3891 — 1

DR DAN WELD
DEPT OF COMPUTER SCIENCE & ENG
MAIL STOP FR-35
UNIVERSITY OF WASHINGTON
SEATTLE WA 98195 — 1

MR JOE ROBERTS
ISX CORPORATION
4301 N FAIRFAX DRIVE, SUITE 301
ARLINGTON VA 22203 — 1

DR TOM GARVEY
ARPA/ISO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714 — 1

MR JOHN N. ENTZMINGER, JR.
ARPA/DIRO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714 — 1

DIRECTOR
ARPA/ISO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714 — 1

OFFICE OF THE CHIEF OF NAVAL RSCH
ATTN: MR PAUL QUINN
CODE 311
800 N. QUINCY STREET
ARLINGTON VA 22217 — 1

DR JOHN SALASIN 1
DARPA/ITO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714

DR HOWIE SHROBE 1
DARPA/ITO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714

DR HOWARD FRANK 1
DARPA/ITO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714

DR BARRY BOEHM 1
DIR, USC CENTER FOR SW ENGINEERING
COMPUTER SCIENCE DEPT
UNIV OF SOUTHERN CALIFORNIA
LOS ANGELES CA 90089-0781

DR STEVE CROSS 1
CARNEGIE MELLON UNIVERSITY
SCHOOL OF COMPUTER SCIENCE
PITTSBURGH PA 15213-3891

DR MARK MAYBURY 1
MITRE CORPORATION
ADVANCED INFO SYS TECH; G041
BURLINTON ROAD, M/S K-329
BEDFORD MA 01730

MR SCOTT FOUSE 1
ISX
4353 PARK TERRACE DRIVE
WESTLAKE VILLAGE C 91361

MR GARY EDWARDS 1
ISX
433 PARK TERRACE DRIVE
WESTLAKE VILLAGE CA 91361

DR ED WALKER 1
BBN SYSTEMS & TECH CORPORATION
10 MOULTON STREET
CAMBRIDGE MA 02238

DR EDWARD FEIGENBAUM
KOWLEDGE SYSTEMS LAB
STANFORD UNIVERSITY
701 WELCH ROAD, BUILDING C
PALO ALTO CA 94304

1

LEE ERMAN
CIMFLEX TEKNOWLEDGE
1810 EMBACADERO ROAD
P.O. BOX 10119
PALO ALTO CA 94303

1

DR OREN ETZIONI
DEPT OF COMPUTER SCIENCE
UNIVERSITY OF WASHINGTON
SEATTLE WA 98195

1

DR GEORGE FERGUSON
UNIVERSITY OF ROCHESTER
COMPUTER STUDIES BLDG, RM 732
WILSON BLVD
ROCHESTER NY 14627

1

DR STEVE HANKS
DEPT OF COMPUTER SCIENCE & ENG'G
UNIVERSITY OF WASHINGTON
SEATTLE WA 98195

1

MR DON MORROW
BBN SYSTEMS & TECHOLOGIES
101 MOONGLOW DR
BELLEVILLE IL 62221

1

DR CHRISTOPHER OWENS
BBN SYSTEMS & TECHNOLOGIES
10 MOULTON ST
CAMBRIDGE MA 02138

1

DR ADNAN DARWICHE
INFORMATION & DECISION SCIENCES
ROCKWELL INT'L SCIENCE CENTER
1049 CAMINO DOS RIOS
THOUSAND OAKS CA 91360

1

DR JAIME CARBONNEL
THE ROBOTICS INSTITUTE
CARNEGIE MELLON UNIVERSITY
DOHERTY HALL, ROOM 3325
PITTSBURGH PA 15213

DR NORMAN SADEH
THE ROBOTICS INSTITUTE
CARNEGIE MELLON UNIVERSITY
DOHERTY HALL, ROOM 3315
PITTSBURGH PA 15213 — 1

DR JAMES CRAWFORD
CIRL, 1269
UNIVERSITY OF OREGON
EUGENE OR 97403 — 1

DR TAIEB ZNATI
UNIVERSITY OF PITTSBURGH
DEPT OF COMPUTER SCIENCE
PITTSBURGH PA 15260 — 1

DR MARIE DEJARDINS
SRI INTERNATIONAL
333 RAVENSWOOD AVENUE
MENLO PARK CA 94025 — 1

ROBERT J. KRUCHTEN
HQ AMC/SCA
203 W LOSEY ST, SUITE 1016
SCOTT AFB IL 62225-5223 — 1

DR. DAVE GUNNING
DARPA/ISO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714 — 1

MS. LEAH WONG
NCCOSC RDTE DIVISIION
53560 HULL STREET
SAN DIEGO CA 92152-5001 — 1

B. MCMURREY
NCCOSC RDT&E DIVISION
CODE 421
53560 HULL STREET
SAN DIEGO CA 95152-5001 — 1

GINNY ALBERT
LOGICON ITG
2100 WASHINGTON BLVD
ARLINGTON VA 22204 — 1

DAVID HESS 1
SAIC ATLANTIC PROGRAMS
ONE ENTERPRISE PARKWAY, SUITE 370
HAMPTON VA 23666

COL ROBERT PLEBANEK 1
DARPA/ISO
3701 N FAIRFAX DR
ARLINGTON VA 22203

ADAM PEASE 1
TECKNOWLEDGE
1810 EMBARCADERO RD
PALO ALTO CA 94303

JIM SHOOP 1
ISX CORPORATION
4353 PARK TERRACE DR
WESTLAKE VILLAGE CA 91361

DR ROBERT NECHES 1
DARPA/ISO
3701 N FAIRFAX DR
ARLINGTON VA 22203

DAVID SWANSON 1
NRAD
53118 GATCHELL RD
44207 BLDG 600 RM 422C
SAN DIEGO CA 92152

DR STEPHEN WESTFOLD 1
KESTREL INSTITUTE
3260 HILLVIEW AVE
PALO ALTO CA 94304

MAJ TOMMY LANCASTER 1
USTRANSCOM/TCJ5-SC
508 SCOTT DR
SCOTT AFB IL 62225-5357

JAMES APPLEGATE 1
MITRE
EAGLE CENTER 3, SUITE 8
O'FALLON IL 62269

SOFTWARE ENGINEERING INSTITUTE
CARNEGIE MELLON UNIVERSITY
4500 FIFTH AVE
PITTSBURGH PA 15213
1

DIRNSA
R509
9800 SAVAGE RD
FT MEADE MD 20755-6000
1

NSA/CSS
K1
FT MEADE MD 20755-6000
1

DCMAO/WICHITA/GKEP
SUITE B-34
401 N MARKET ST
WICHITA KS 67202-2095
1

PHILLIPS LABORATORY
PL/TL (LIBRARY)
5 WRIGHT STREET
HANSCOM AFB MA 01731-3004
1

THE MITRE CORPORATION
ATTN: E LADURE
D460
202 BURLINGTON RD
BEDFORD MA 01732
1

OUSD (P)/DTSA/DUTD
ATTN: PATRICK G. SULLIVAN, JR
400 ARMY NAVY DR
SUITE 300
ARLINGTON VA 22202
1

DR. ROBERT PARKER
DARPA/ITO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714
1

DR. GARY KOOB
DARPA/ITO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714
1

DR. ROBERT LUCAS
DARPA/ITO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714 1

DR. DAVID GUNNING
DARPA/ITO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714 1

AFIT ACADEMIC LIBRARY/LDEE
2950 P STREET
AREA B, BLDG 642
WRIGHT-PATTERSON AFB OH 45433-7765 1

US ARMY STRATEGIC DEFENSE COMMAND
CSSD-IM-PA
P.O. BOX 1500
HUNTSVILLE AL 35807-3801 1

NAVAL AIR WARFARE CENTER
6000 E. 21ST STREET
INDIANAPOLIS IN 46219-2189 1

COMMANDER, TECHNICAL LIBRARY
474700D/C0223
NAVAIRWARCENWPNDIV
1 ADMINISTRATION CIRCLE
CHINA LAKE CA 93555-6001 1

CDR, US ARMY MISSILE COMMAND
RSIC, BLDG. 4484
AMSMI-RD-CS-R, DOCS
REDSTONE ARSENAL AL 35898-5241 1

REPORT COLLECTION, CIC-14
MS P364
LOS ALAMOS NATIONAL LIBRARY
LOS ALAMOS NM 87545 1

AIR WEATHER SERVICE TECHNICAL
LIBRARY (FL 4414)
859 BUCHANAN STREET
SCOTT AFB IL 62225-5118 1

AFIWC/MSO
102 HALL BLVD, STE 315
SAN ANTONIO TX 78243-6016 1

PHILLIPS LABORATORY
PL/TL (LIBRARY)
5 WRIGHT STREET
HANSCOM AFB MA 01731-3004 1

AEDC LIBRARY
TECHNICAL REPORTS FILE
100 KINDEL DRIVE, SUSITE C211
ARNOLD AFT TN 37389-3211 1

SPACE & NAVAL WARFARE SYSTEMS
COMMAND (PMW 178-1)
2451 CRYSTAL DRIVE
ARLINGTON VA 22245-5200 1

MISSION OF ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Material Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.