Integrating MUMPS and SQL

A System Architect's Overview

W. Lyle Schofield

Abstract

Many systems developers are currently trying to integrate MUMPS technology to non-MUMPS technology. Relational databases represent unique challenges due to the structural differences between relational (SQL) and hierarchical (M/MUMPS) structures, and most users desire to tightly couple the databases. This document presents an overview of this topic, focusing on architecture alternatives and general decisions to be made by the designers.

Introduction

There was a time when many of us enjoyed the luxury of working in environments where applications ran on a single processor with all the source code in a common language. Over time we watched this genetically pure system invaded by purchased executable files with no source code, split into multiple processors, then clustered and LANed into multiple operating systems. Most recently, end users have replaced their terminals with workstations so that they could do word processing while entering data, and are now demanding integration with their departmentally developed Access applications. As we look into the future of object oriented, web-enabled, network computers we begin to wonder about the place for MUMPS in this next generation of applications.

Regardless of protocols and methodologies, source code is source code and there continues to be appropriateness for MUMPS development. Obviously, legacy system support and legacy system integration will continue to occupy programmers' time. MUMPS will also enjoy its traditional advantages of modest hardware requirements for multi-threaded high volume processing. As "4GL" technology becomes more common and development tools more graphical, MUMPS also carves out a new niche as a development environment that supports data structures as well as low level device support in a single environment[1]. And of course, MUMPS evolves and features yet unknown will be added to the language.

So life with MUMPS continues, but the environment is now the heterogeneous client/server model common in both large and small organizations. The challenge is to share MUMPS data with non-MUMPS data, most commonly SQL databases sold by Oracle, Sybase, Informix, and others. SQL helps simplify all these different products by providing a somewhat standard programming language[2]. Several MUMPS features and add-on products provide connectivity paths. But the design process for a project linking these environments has many alternatives, and live installations are trade secrets or small in scale.

This article hopes to outline alternatives for linking MUMPS to SQL and some major considerations for application designers.

User Requirements

Any application development process must begin with the question of "what is the problem we are trying to solve?" The answer needs to be a specific statement of work, and should avoid the vague need to make a "better" system on a "mainstream" product. Organizations that frame problems in this way are doomed to Word and Excel macros as the answer to all their problems.

The job of the designer is to coach users into stating their problems with specific references to organizational challenges: a need to move data electronically, provide less data entry time, allow data analysis in a more flexible way.

Ultimately, the linking of MUMPS and SQL is an application development project to integrate. The business needs driving this tend to group into the following categories:

- Provide A Reporting System
- Integrate Data from Applications
- Move Data (One Way) Between Systems

Provide A Reporting System

Any organization that is proficient at collecting data will ultimately want to look at all the data they have collected. Many applications provide tools for this, and at some point tools become outdated or not as flexible as needed.

There are several ways to solve this problem. The most obvious, especially for a homogeneous MUMPS shop, would be to install a MUMPS-based reporting tool. There are several, some even using SQL syntax, and most offer high performance and can be supported by existing programming staff.

Sometimes resources consumed by report generation can

impact the operational needs of a system. You can upgrade your system, or install a second system to provide resources for reporting, removing this demand from the operational system. A brute force approach would be to install a second MUMPS system, and copy your operational back-up files to the reporting system. This is an easy snapshot of the system from some point in time, and the development effort to implement this tends to be little or none.

If a separate reporting system is being set up, it does not have to be MUMPS-based. It could be based on some other database technology, like SQL, using some other reporting tools, like the dozens of SQL-based tools currently available. There is additional development effort to export and import the data, and to define data structures for holding data on the SQL system.

Perhaps the need to improve reporting is due to data existing on two or more hosts, some MUMPS, some SQL. The reporting system will now need to talk to both technologies, and the need to integrate MUMPS and SQL is obvious. The reporting function could be happening in MUMPS, with MUMPS reading through to SQL, or the reporting function could be happening in SQL, with SQL reading through to MUMPS.

As the environment grows, SQL could be the common protocol talking to MUMPS and SQL. SQL vendors are currently providing many new workstation-based graphical tools, providing powerful analysis as databases grow past terabyte sizes.

Integrate Data Between Applications

In many organizations, best-of-breed applications have been selected to solve specific enterprise or department needs. Some might be MUMPS-based, some might be SQL based. At some point it may be a requirement to move data electronically between systems.

For example, patient demographic information may be represented on multiple systems. If an address change is made on one system it would be nice to migrate that change to all other systems containing this information[3].

Users will have to decide how real time this information exchange needs to be. The solution could be *interface*, immediate movement of data between systems, or *interchange*, moving data in a timed batch.

Move Data (One Way) Between Systems

There may only be a need to move the data in one direction. Many organizations are deploying data warehouses for archival and analysis work. In this application data will be sent from MUMPS or SQL systems to another system (MUMPS or SQL). The designers of the data warehouse will design their own data structures optimized for the warehouse function[4]. Data will be packaged up and sent to the warehouse, through a batch transfer or possibly a real time interface. Data is not created in the data warehouse or sent back to other systems.

A one-way interface may also be needed as programmers prepare to migrate data to a newly engineered application. An organization may decide to move from one technology to another based on long-term strategic plans. An interface is designed to run once as users are migrated to the new version of the application.

Integration Points

We have discussed why we need to integrate. Before we get into the details of how, we need to show where we can integrate.

First, we will provide the following diagram to represent any system.

Fig. 1 System Layers

Any system can be conceptualized as functional layers. I have chosen four for this discussion.

The *network layer* provides physical and logical connectivity between computers.

The *operating system* manages the resources of that computer. It manages tasks, memory, provides access to the network and other hardware components, and other administrative trivia like keeping track of the time.

The *database layer* stores data. It allows multiple processes to save data, examine data, and allows programmers to define the structure of this data. This would be the global variable component of a MUMPS system, and the SQL component of a 4GL system.

The *application logic* layer defines how elements of data interact with users. In a MUMPS system this would be programming that defines types of data, edit checks on that data, and other logic that makes up the application. In an SQL system this is the stored procedures, triggers, and data types that define the relationships between the data.

Any of these layers can be integrated with their associate on another machine, as in the diagram on page 40. Network integration is the only function of the network layer. Physical connectivity is done through compatible wiring and connectors, and protocols define how the electronic squirting around the wires are interpreted as data and addresses. Developers assume that the networks will provide needed services and usually don't write applications with specific

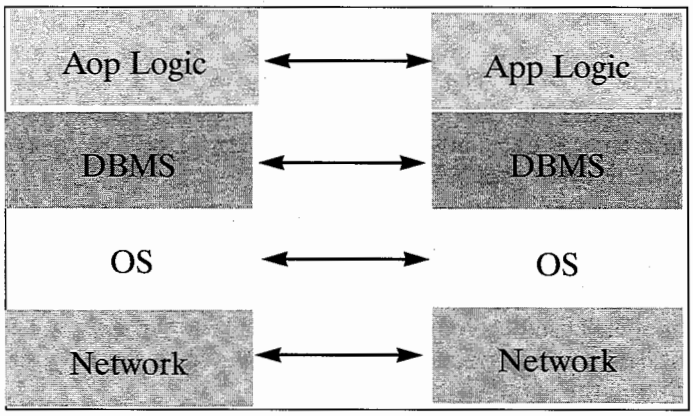

Fig. 2 Integration Layers

networks in mind. While there are interesting things happening on this level, they have little impact on the design of the application. We will assume that a network engineer will write an article explaining this layer in great detail.

Operating system layers are generally not integrated. In a clustered environment, the operating system is very tightly integrated with another system, with different members of the cluster sharing information on processes and resources. Again, this is not of much interest to the application designer since we will assume that the operating system will work as advertised and we will refrain from the desire to change programming of the operating system.

The database layer does allow integration between systems, and this is of interest to the application designer. MUMPS can integrate with SQL on this layer through products that allow MUMPS data to be defined with a relational view, then share this information with other SQL engines.

The application layer also allows integration. Transactions can be defined to the different systems which can be used as integration points. These transactions may be well-known such as HL7, ASTM, and other vendor-independent protocols, or custom for the specific application. There is also a product niche referred to as integration engines tool kits and CASE tools that facilitate integration on this level.

The database and application integration layer will be examined in more detail, with additional discussion on middleware.

Database Layer

Database integration has been accomplished through a number of methods.

MUMPS-to-MUMPS integration can be easily achieved through proprietary vendor links, or through the OMI non-proprietary protocol. This allows high performance clustering of MUMPS resources as well as application integration.

SQL-to-SQL integration has been accomplished through similar methods. Vendors offer proprietary links between separate systems running their products, and provide non-proprietary links through ODBC or similar protocols to share data between different vendors' products.

To get MUMPS integrated with SQL systems, there needs to be some common denominator. Either SQL needs to understand OMI, or MUMPS needs to understand ODBC. Given the ratio of OMI systems to ODBC systems it is not surprising that the latter approach is available.

The main obstacle to having MUMPS and SQL share data is the structural differences in how they store data. MUMPS uses an hierarchical storage technique with its global variable arrays, while SQL uses the relational model of two-dimensional tables containing columns and rows. Regardless of vendor product selected there is a need to map MUMPS data structures to the relational model. Each MUMPS-to-SQL product contains a tool to do this, where a programmer who understands MUMPS structures and also understands the relational model creates a map of global variables to SQL structures[5].

Once the model has been created it can be used by developers on the MUMPS system for application and report development. It can also be used by an interface to serve SQL data to the rest of the network. Commonly, ODBC is used as a transport for this data, although some products support specific vendor network proprietary interfaces, such as Oracle's *Oracle Call Interface* (OCI).

When MUMPS has been mapped and can serve its data relationally on the network, it now looks like any other SQL database to ODBC. To the database clients, MUMPS provides data just like Oracle, Microsoft's SQL Server, or Microsoft Access. This opens up many reporting tool possibilities for the client workstations. Users can use Microsoft Access for a user-friendly ad hoc reporting tool. MUMPS data can be used by Oracle Explorer and data mining tools. Programmers not familiar with MUMPS can now pull data off the systems using SQL tools they are comfortable with. And MUMPS programmers now have access to non-MUMPS data floating around the organization.

Application Layer

It is possible to integrate systems on the application layer, where programming is developed to import and export the data across some agreed protocol. This can be done through a batch file transfer process, or through a real time link. Integration can be achieved regardless of operating system or database technology as long as the developers are able to agree to a protocol and implement it.

A batch transfer is an obvious way to implement this link. Assuming that there is not a need for real time integration[6] a file format is agreed to by all systems sharing data. Developers write code to export to the file format and import into the database as needed. It is also possible to integrate this way with applications

that do not support any integration protocols, such as using a file export utility from a desktop application.

When immediate integration is a requirement, applications will need to have an integration point created for them. These take the form of a *remote procedure call* (RPC) which is a "published" interaction with the application. This could be a simple call posting a query and receiving acknowledgment, or a more complex interaction that posts a database change and follow-up messages.

This type of approach allows programmers to centralize database changes through common transaction processing, using the same logic for database changes coming from the network as those coming from locally attached users. There is also an ability to provide higher performance than a database-to-database binding, since the RPC would be a terse transaction that could accomplish many database changes.

Vendors of integration engines are using this approach in many cases. Integration Engines come in two flavors: a software flavor, such as ISG's *CorVision*, or a hardware flavor, such as STC's *DataGate*. The software version is more of a developer's tool kit to provide application integration services with an assumption that there is already a physical connectivity path. The hardware version sells a box to plug in multiple networks providing the physical path, with the software version added to perform the application integration. Note that the hardware version might be overkill if there is already physical connectivity.

The next diagram shows how an integration engine might attach two different LANs where there are currently incompatible application interfaces.

The process of designing an application interface is much like the process of designing a user interface. Designers and users need to come together to define what data is passed through the interface and how the interface is responded to. Based on those specifications, the "client" and "server" software is written on both sides of the interface.

This type of interface also has interesting ramifications for future system modifications. Much like the process of defining the database integration using a relational data map provides the ability to add any client on the network that is ODBC-capable, the development of the application interface allows any client following the RPC protocol to interact with the interface. For example, an RPC could be developed that allows an SQL system and a MUMPS system to share data. A programmer could also write a Visual Basic application that acts as an interface client, providing a true client/server version of the application.

In the figure 3 above I specifically list that applications are running HL7 and "not HL7" to highlight the application integration of the engine. HL7 is a vendor-independent interface for health care. It defines typical information used

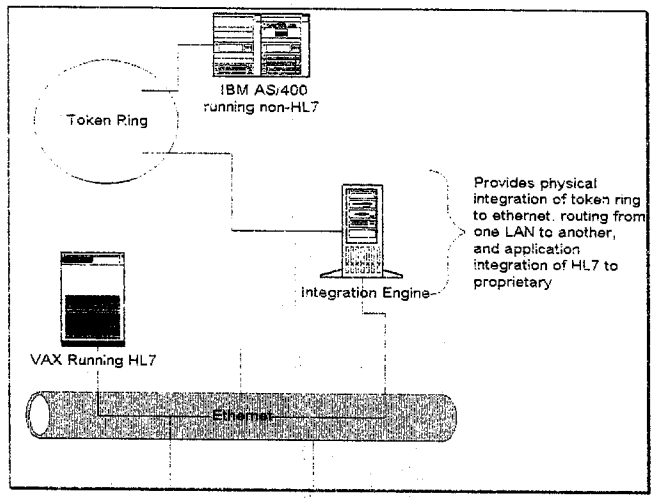

Fig. 3 Integration Engine Providing Application Integration

in hospital information systems, such as patient demographic information and patient discharge information. There are other such interfaces defined, such as ASTM for lab equipment and ANSI X.12 for health care financial information. While it is hard for the interfaces to be defined as rapidly as the changes in systems and data evolve, there are obvious application interface advantages for using these standards. A vendor has the ability to define one interface and support it, and not worry about custom interfaces to other systems. Other developers can take advantage of these interfaces for supporting legacy system integration, or supporting integration to home-grown systems.

As industry standard interfaces develop the need for integration on the application level is "built in."

Object Brokering

Looking toward the future, there is a lot of movement toward *Object Orientation* (OO) for development tools and as a development methodology. The complexity of merging this change with client/server development has led to the definition of *network objects*, or the ability to provide data resources on a network in an object oriented fashion.

There are a number of products on the market that allow client/server development using object oriented methodologies. Most of them provide object services between the client and a *middleware* server, and the middleware server talks to a DBMS server on the back end running a relational engine. This allows object based development for the client, a middleware component that serves business logic, and deployment using a three-tier architecture. There are several characteristics of this approach.

Object Development The goal of object orientation is to allow easier re-use of source code. While a detailed explanation of OO technology is beyond the scope of this article, OO shifts

the focus of developer from huge code listing of logic to codelettes that define the unique properties of the object at hand to other objects. The object at hand inherits the properties of objects it is inside of. When we define the date-of-service object as being inside the encounter, the date-of-service object inherits the pointers to patients and facilities defined for the encounter.

The re-use of code is a part of the development environment, and not a separate library to be found and attached. The general weakness of code libraries in other environments is that it is sometimes not easy to find a function that has already been defined. The previously completed definitions in an object environment are part of the application that can be seen on line with the development tools.

Fault Tolerance Most implementations of object middleware support multiple middleware servers with object replication. In the event that a middleware server goes down the clients shift to another server.

Application Partitioning In a multi-tier deployment, developers using non-object tools often have to decide which services run on which tiers. The act of deciding which parts of the application (presentation, business logic, database, etc.) run on which layer is often referred to as *partitioning* the application. Making a wrong call can result in poor application performance. Often, it is hard to predict the future bottlenecks of a complex client/server application once network equipment and traffic are thrown into the equation.

High end object/client tools may have a *dynamic partitioning* feature. This allows the network administrator to decide which parts of the application run on the servers and which parts run on the client. This ability to tune on the fly, combined with the ability to throw additional middleware and database servers into the system, allow powerful scaling and tuning after deployment.

While these features show the direction software development is headed in enterprise-class deployment, there are some issues. The most obvious are the costs of deployment. With training, higher level hardware, additional servers and server software, and client license costs, projections of more than $1000 per client for software and upgrades are not unusual. More importantly, most of these solutions are proprietary to their vendors, locking developers down the vendor's path for the time being. Given the small market penetration of these tools at this time, early adopters may be living with "bleeding edge" issues depending on selection.

There is movement to provide industry standards to network objects. Two initiatives, Common Object Request Broker Architecture (CORBA) from a consortium of vendors and Common Object Model (COM) from Microsoft, are trying to define the ability to query the network for object information. This will allow application developers to write applications without any regard to their location on physical clients and servers, and the network protocols will find the service and provide it.

To relate this back to MUMPS and SQL integration, SQL vendors will have integration to this environment as they aggressively move to include object tools with their products, and the object middleware vendors have already implemented into SQL database engines. MUMPS could provide SQL services through the ODBC binding, although there does not seem to be much value added to this arrangement. As an alternative, MUMPS could participate in the CORBA and COM initiatives. This implies that object features are rolled into the MUMPS products. We shall see where MUMPS vendors go with this.

Additionally, a third party could write a CORBA server for MUMPS. Similar to the SQL translator that sits on MUMPS, an object and object broker could be written to sit on top of MUMPS.

Engineering Decisions

After understanding implementation paths for integrating MUMPS to SQL, we need to list the pros and cons of the different approaches.

The single biggest cause of failure in projects like this is not having a full understanding of application requirements. It is important to understand the needed integration, as well as presenting the true costs and time frames for implementing the solution.

Perhaps the biggest issue to resolve is the need for real time integration. There can be a large difference in cost and time for providing an import/export utility versus a real time network transaction engine. If data needs to move, but can wait anywhere from five minutes to a day, a batch process can usually be developed more easily than an SQL database map. With the ease with which files can be moved between systems and easy access to scripting tools to automate this process, moving import files between systems is not a bad alternative.

The second question to answer is "what are the existing interfaces"? Many applications already provide some electronic connections. These were either developed to support conversion to the system when installed, or to support electronic data interchange or other integration. For example, if the vendor has already provided an HL7 interface, take advantage of it. Either write an HL7 interface for the second system, employ an integration engine to attach the HL7 interface to another one, or attach the two HL7 interfaces that were there all along! If someone else has written the code for you, use it.

After answering the questions and documenting the design, you are faced with the decision of whether to integrate on the database layer or the application layer.

The database layer has some obvious attractions. MUMPS vendors support tools to let you develop the relational view of your application. There are a bazillion and three ODBC-compliant tools and applications (based on my last survey), many of which users are familiar. And connectivity to the SQL engine gives you access to a host of modern, visual development tools, including those that do not have visual as part of their names.

There is one major disadvantage to the database integration method. As seen earlier in our diagram of database integration, the databases are talking to each other. But when application users are shown we see that they attach to the application logic layer. The implications are that application logic must be kept in synch between the two systems.

For example, when MUMPS wants a piece of data from the SQL machine, such as a person's physician, it will ask the SQL machine and receive a reply. But what happens when the MUMPS machine tells the SQL machine to change the physician to a specific value? The SQL machine will have to perform the edit checks to make sure this is a valid selection. The application logic on the SQL machine will have to exist in the MUMPS machine to make sure the database logical integrity is insured. This is a similar issue to client/server applications where the client is "fat," meaning that the client contains business logic such as edit checks and filing rules. If the application changes, the updated software must be distributed to all the clients in order for them to work correctly. If you pursue a strategy of database integration you will need to make sure the relational table map on the MUMPS system reflects any changes made in the database. You may also need to make sure SQL programming changes are made if the MUMPS system has logic changes made.

There are work-arounds for this issue. One of the databases could be made "dumb," a physical repository with no checks on certain areas of the database. Or users could be instructed to enter information on one machine and not the other.

If the purpose of integration is for reporting only, this issue goes away. The server (MUMPS or SQL) is integrated on the database level, but configured so that the data is *read only*. Users can pull data from the machine and the data will not be harmed by incorrect business logic during filing.

Based on requirements, an application level integration may be more attractive.

If there is an existing interface on one system or the other, write to that interface. An integration engine will shorten development time with the tradeoff being additional cost and management. Remember that with some integration engines you are paying for features (support for token ring, for example) that may not be needed. At the same time, you may also receive interface definitions already defined by the vendor.
During the design process, examine your existing application for services that could be utilized by another interface. Are the parts of your application that make database changes written in a clean, transaction processing-type convention? This makes it easy to attach another interface, and also simplifies maintenance.

If you are writing a background process that will provide interface services, what are the expected volumes? If this will be a high volume service make sure you plan for multiple processes. Also make sure that you provide auditing of the activity of the processes. If it difficult to debug an application that only reads and writes to a network port, trap as much information as you can afford to.

As you compare creating a database link to an application-link, remember that there are other advantages to the work of building a relational map along with the integration.

The relational mapping tools will provide access to new development and reporting tools. Both the KB/SQL-based products from KB Systems, Micronetics, and Oracle as well as the M/SQL product from InterSystems have an SQL reporting tool attached. As mentioned earlier, this can be a solution for ad hoc reporting as well as production reports created by programmers. Additionally, the M/SQL product is integrated with a form generator and routine editor to provide SQL tools for application developers.

The exercise of mapping MUMPS data relationally will provide experience porting the application to an SQL system. This may help in an effort to move applications to this platform, or may provide evidence that redeveloping the application in SQL does not offer significant advantages.

As a final comment, please note that many off-the-shelf integration products may not have much of a track record, or may have limitations on how high you can scale them. Make sure you check on the product's installation history and compare this to the expected transaction volume for your site.

Conclusion

While this article was kept at a high level, it is hoped that it has provided an overview of issues related to integrating MUMPS with SQL systems.

A MUMPS-to-SQL integration project is like any other project; good requirements definition and planning will lead to success. Also remember that there are many alternatives to linking MUMPS to SQL. List each alternative that meets the requirements, list the costs associated with each possibility, and perform the cost/benefit analysis. M

REFERENCES

1. 3GL platforms often call into 4GL environments using an API to manipulate and examine data structures. Similarly, 4GL programming environments often call out to 3GL code to talk to hardware and perform CPU intensive tasks. Many components of enterprise

applications such as interfaces and file import/export, can easily be implemented in MUMPS with quick access to data structures as well as simple commands (OPEN, *USE)* to control peripherals.

2. Of course, many vendors have their own extensions and flavors of the SQL standard. From a programmer's standpoint this is no more annoying than different versions of MUMPS. Systems Management issues are another story (skills do not easily translate between vendors' products) and visual development tools that extend basic SQL functionality vary widely.

3. In some industries vendors are addressing this through the definition of independent interface transactions. This will be discussed later.

4. A data warehouse is simply an application after all, with its own design criteria.

5. This is not often a trivial job. A typical MUMPS application will contain repeating structures inside repeating structures. A literal translation of this often results in sub-optimal relational definitions.

6. Although if the batch can be processed fast enough, it can be close to a real-time link.

7. This is probably part of the attraction of the "thin" client or "network computer" (NC), a PC running a web browser and JAVA or ActiveX as client software. Software, while not always running on the server, is always fetched from the server. This makes for a simple software distribution model in a client/server world.

Lyle Schofield is VP of Consulting Services at Sentient Systems, Inc., He can be reached at:
lyle@sentientsystems.com.

Advertiser Index

We appreciate these sponsors of the July issue and all the companies who support the M community through their commitment to excellence.

This index appears as a service to our readers. The publisher does not assume any liability for errors or omissions.

Announcing: A New Distance Learning Course on M Available On the Internet: A Trial Offering

Courses on M are usually available at scheduled times and scheduled places. These schedules do not often fit precisely with the needs of the student. In an effort to increase the opportunities for learning about M, the University Extension Division of the University of California, Davis, is planning to offer a 10-week course in beginning and intermediate M programming on the Internet. This class will be "taught" by Dick Walters, who will use materials from courses he teaches at UC Davis, tutorials offered at the MTA annual meetings over the past few years, and material from the revised version of The *ABCs of MUMPS*, due to appear in early 1997 under a different title.

The class will begin with basic concepts of programming, and move into features that make M such an effective tool for text manipulation and data storage. Students will learn how to write useful programs in M, and move on to questions of modular design, package development, and advanced features of M.

Course materials will be made available in part by a mail shipment of exercises and other reference materials, and in part through resources on the Internet. Students enrolled in the course will receive an account on a class computer and be given access to teaching materials stored on-line and accessible through the Internet and World Wide Web.

An important component of the learning process will be use of a new communication tool: the Remote Technical Assistance (RTA) package. Using this package, students will be able to send both deferred complex messages (including screen snapshots, file attachments and sound clips) which can be annotated by the instructor and returned to the student. Live interaction will also be possible at times when both student and instructors are both on line, with whiteboarding, sound messaging and WWW URLs invoked by the instructor on the student's screen. Group discussions between students and instructional staff will also be provided. The RTA package is operational on Macintosh computers and will soon be running on both PC and UNIX client systems. A UNIX server is used to store class lists, teaching resources, messages, dialogs and other results of interactive conversations.

This course represents something of a new experiment in several respects, since neither University Extension nor the instructor have attempted a course with quite this format before. For that reason, the first group of students signing up for the course will be able to take it at a very low cost (amount yet to be determined). Their experience will be used to create a course that will be made available over the Internet on a self-paced basis. For further details, send email to: *walters@cs.ucdavis.edu*

Dick Walters, Department of Computer Science
University of California
Davis, CA 95616
fax (916) 752-4767
(no voice messages please)