

Clients <-> Agent <-> Servers: Accessing Distributed, Heterogeneous Databases

Sigurd W. Hermansen and Greg Cox, Westat

ABSTRACT

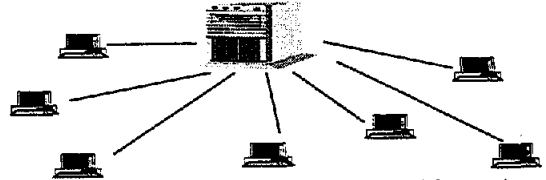
Databases in the future will certainly become more widely distributed across applications and platforms. Even now we are seeing on intranets and the Internet a surge in database implementations that cross DBMS and system boundaries. That trend seems likely to accelerate. Concurrently, the boundaries of database systems are being stretched to encompass multiple forms of data dictionaries, different data types and objects, and new ideas about the relations among data objects. Distributed and heterogeneous databases open up new possibilities for systems, but present new challenges as well. Builders of database access systems are relying more and more on middleware solutions to the problem of combining data. In this presentation, we suggest a scheme that links database clients and servers under the control of an agent program. As a simple example, we present a MS Access form that builds a SAS SQL query for a networked client PC, directs the query to a SAS database on a Unix server under ODBC protocol, and displays the yield of the query on the PC as an MS Access table. This example provides a springboard for a discussion of the advantages and difficulties inherent in clients<->agent<->servers designs for distributed databases.

Those of us who have survived a few shifts from centralized databases to client/server configurations understand at once the advantages of distributed, heterogeneous databases. Those who have been spared that experience so far may need some convincing. A bit of history may help put all of us on the same page.

The Nervous System Model of a Database

Up to a decade or so ago, most database developers thought of computer systems as an artificial nervous system. This model appeared frequently in my system documentation as something along the lines of the diagram that appears below.

Nervous System Model

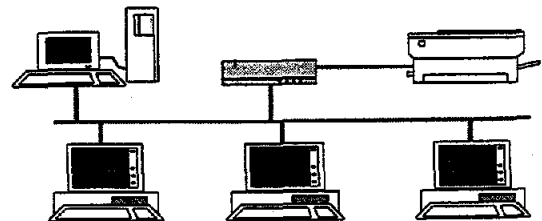


The CPU "brain" connected through a hierarchy of nodes to terminal points (called, not surprisingly, "terminals"). A terminal "sensed" data, encoded them, and transmitted them up the hierarchy to the CPU. The CPU acted on the data it received and transmitted data back to the terminal. A user working at a terminal had, in effect, a keyboard connected to a CPU by a very long wire.

A Circulatory System Model

About the time that computer boards shrank to chip size, terminal manufacturers began making smarter terminals. (Does anyone else remember the winking owl in the Wyse terminal logo?) Pretty soon users began using PC's with their own little CPU's and modems to connect to mainframe databases. First came spreadsheets and then PC databases. The distributed databases that were becoming more common did not fit the nervous system model. Before long users were asking for extracts from mainframe databases for their local databases, and centralized databases were begging for on-line customers. The networked databases that evolved looked more like a blood circulatory system (below) than a nervous system.

Circulatory System Model



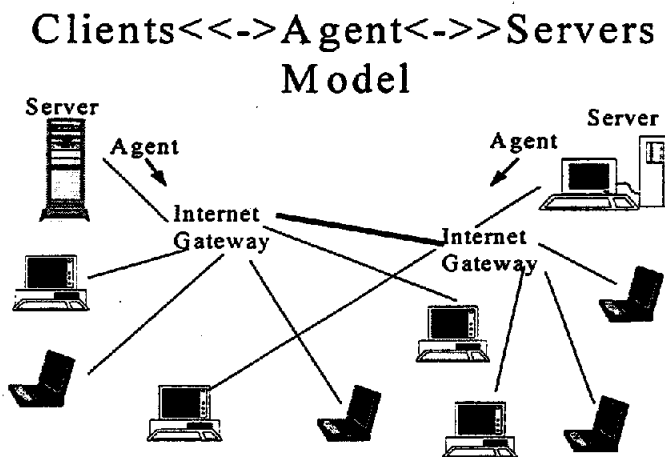
Computers connected to the network at various points and released packets of data, each with a target address, into a data stream that circulated to other computers. Each computer on the network selected the packets addressed to it and filtered out the others. The Unix telnet and ftp protocols helped extend access to these circulatory network systems to everything from classic mainframe to PC. Multiple

clients/one server systems further extended these circulatory networks to interchanges of queries and transactions across different platforms.

Clients <-> Agent <-> Servers

With the arrival of the Internet, database configurations have begun to evolve further. By Year 2000, databases systems will extend across all of the old platform and network boundaries. Multiple clients will have access to multiple servers. Network connections will eventually retreat to the background to such an extent that users will only understand them metaphorically, much as desktop users see folders instead of directories and files. We are already seeing the first wave of the "intelligent agents" that users will send searching for information. These agents fit into the gaps between clients and servers. In the clients <-> agent <-> servers metaphor, an agent understands the users' languages for expressing requirements for data and translates them into forms that servers can understand; the agent also knows how to present the servers' findings into forms the clients can understand.

The following diagram shows an outline of a clients <-> agent <-> servers model.



How It Might Work

The Agent in the Clients <-> Agent <-> Servers model has to perform a number of basic functions. But what it does not need to do has as much importance as what it needs to do. A number of vendors offer total solutions. We regard them with suspicion. We are looking for a good assembly of parts, not whatever a vendor happens to own. For example, a vendor may offer a product that accesses the contents of Web sites. A number of GUI development systems facilitate creating forms and

linking them to databases. A number of protocols support database access. In particular, the ODBC protocol allows clients to pass SQL queries through a network to a server and to receive the results as a table. These and other products already exist.

The Agent has to fit within a typical client/server configuration. An effective and commercially effective agent program, the Adobe Acrobat Reader, has a similar role in document viewing systems. Users can link to a Web page, find the correct document reference, and copy a .PDF document and the appropriate Acrobat Reader from the Web site. Adobe has different Acrobat readers for MS Windows, Unix, and Macintosh systems. The Acrobat Reader displays text and graphics in the many different formats used by desktop publishing and graphing programs. It accompanies the document object on the server and translates it into a display on the client.

The agent we have in mind has a similar role in the process of requesting data from databases on servers.

Like the Acrobat Reader, it is a program that would run on different platforms. It determines the attributes of the client and matches them to the appropriate object. It helps the client copy the object and start it running. The object transfers metadata from databases on the servers and presents a blank form on the screen of the client's machine. The user views the metadata and enters the details of a data request. The object uses the data request to write a query program that the servers will understand. It uses a standard protocol to pass the query program through the network to the servers. The servers process the query program and return the result. The client displays the results as a table. The user can then print the results or save the table.

A Working Example (Still under Construction)

As a proof of concept, consider the Specimen Test database being developed at the Medical Coordinating Center (Westat) for the National Heart Lung and Blood Institute's Retrovirus Epidemiologic Donor Study (REDS). The agent, an MS Access object containing a program module written in Access Basic, links under an ODBC protocol to a SAS library on a Unix machine. It extracts all distinct values (domains) of selected variables in the SAS database and displays as selection lists on the form shown below.

Form: REDS Special Testing

REDS SPECIAL TESTING

Select/Deselect A Center:

☐ ARC Chesapeake	☐ ARC So. California	☐ Oklahoma Blood Institute
☐ ARC Detroit	☐ Irwin Memorial Blood Center	

Specimen:	Donor:	Subject:	Lab ID:	Test: PCR
Disease:	Panel:	Repository:	Result:	Laboratory:
Draw Date: From: To:		Order Date: From: To:		Test Date: From: To:

Select/Enter Search Criteria

Passed-in SQL Code:

```
create table spt.cox143 as select o.*,p.descript,c.descript as cdescript from spt.order
as o left join spt.panel as p on (p.panel=o.panel) left join spt.comment as c on
(c.comment=o.comment)
where ( o.center = '1' or o.center = '2' or o.center = '3' or o.center = '4' or o.center = '5');
```

Passed-in SQL Code:

```
create table spt.cox243 as select l.center, l.specimen, l.labid, l.lab,l.reposito,t.testdate,
t.test,t.result,lab.laborato as laborato from spt.labid as l left join spt.test as t on (t.lab_id=l.labid)
left join spt.lab as lab on (lab.lab=l.lab) where (t.test= 'PCR');
```

Passed-in SQL Code:

```
create table spt.cox343 as select s.center, s.specimen,s.drawdate,d.donor,s2.subject from
spt.specimen as s left join spt.donor as d on (d.center=s.center and d.specimen=s.specimen)
left join spt.subject
as s2 on (s2.center=s.center and s2.specimen=s.specimen) where ( s.center = '1' or s.center
= '2' or
s.center = '3' or s.center = '4' or s.center = '5');
```

Converging Query:

```
create table spt.coxg as select s.center as
center,s.specimen,s.drawdate,s.donor,s.subject,l.labid,
l.lab,l.reposito,l.testdate,l.test,l.result,l.laborato,o.panel,
o.orderdat,o.disease,o.comment,o.cdescript,o.descript
from spt.cox343 as s left join spt.cox243 as l on (l.center=s.center and l.specimen=s.specimen)
left join spt.cox143
as o on (o.center=s.center and o.specimen=s.specimen) where (( s.center = '1' or s.center = '2'
or s.center = '3' or
s.center = '4' or s.center = '5')) and (test='PCR');
```

Form: special result table form

Result Table one record at a Time

Specimen:	00231115	Center:	
Subject:	PDT-033444	Donor:	0000012537
Repository:		Drawdate:	5/2/91
LabID:	W0622	Laboratory:	SRA
Testdate:	5/24/91	Orderdate:	2/14/95
Test:	PCR	Result:	NEG
Disease:			
Panel:	EDXSRA INVESTIGATING HTLV INDE TERMINATES FOR		
Comment:	test comment field		

Previous Record First Record Print Report EXIT

Next Record Last Record Save Report >> R-reported

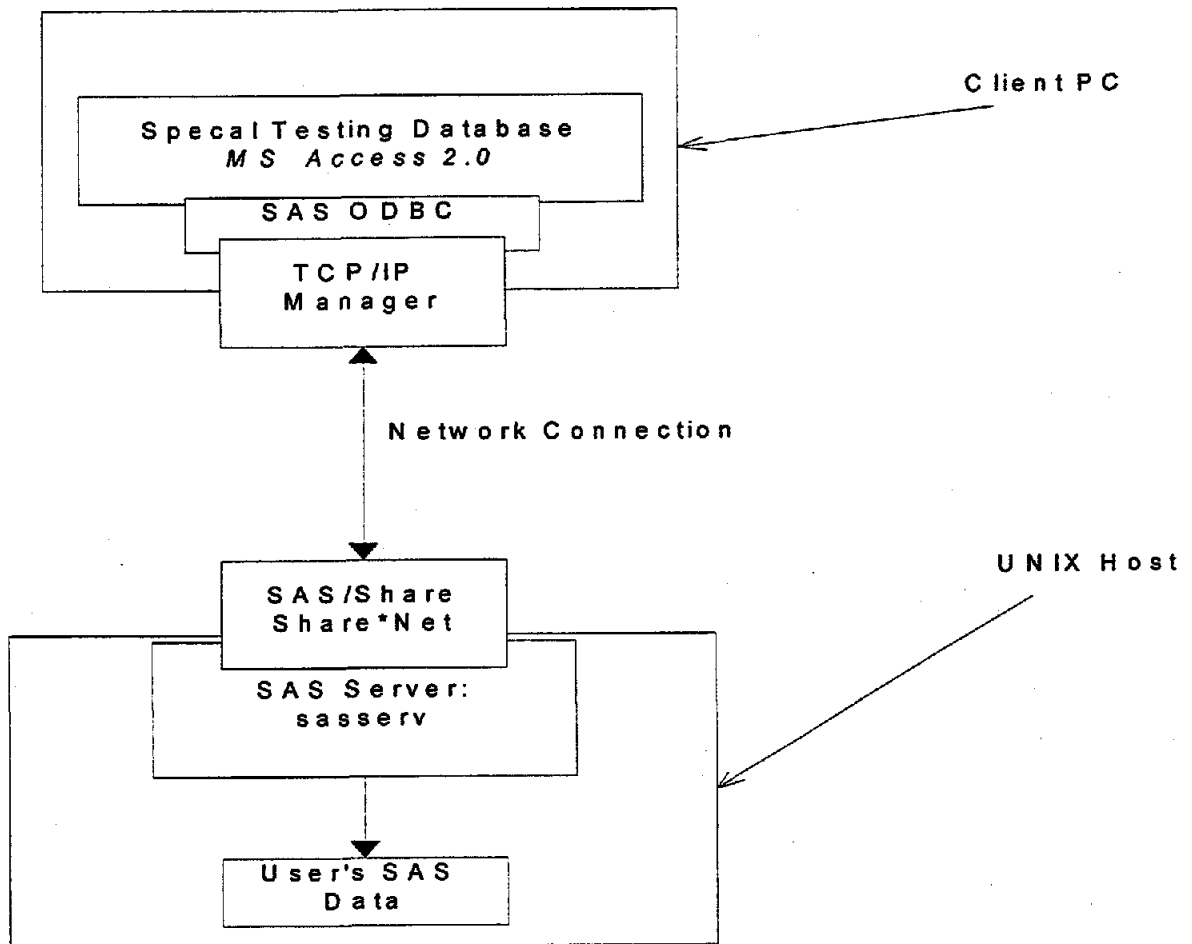
There are 2108 records Query Time: 15 seconds

The form includes labelled slots for search criteria, lists from which to choose values, and instructions for using the form to request data from the database. The layout of the form and the labels orient the user and display the range of choices. The user does not have to understand the details of the database scheme, the implementation, or the platform to request data from the database. Nor does the user have to know where the data actually reside or how the agent locates them. (This gives the database administrator the option to reorganize and relocate data when necessary).

When the user completes and submits the request, the agent embeds the search criteria in a SQL statement written in the dialect(s) of the servers. The MS Windows ODBC configuration passes the SQL statement through to an IP address of a SAS server running SAS Share and SAS SHARE*NET. The "passed-in" SAS . SQL code appears above. The server is listening at that address for SQL queries embedded in ODBC scripts. It executes the query as a Unix process. The process transfers the yield of the query back to the IP address of the client, where the client displays the data as an MS Access table. In the example above, a special form displays the result Table one record at a Time. The diagram that appears below shows the relation between Network Communication Managers and the client and servers.

This example falls several elements short of a complete package. It does not include the agent program on the server that manages setting up clients so they can display the query form; public use query form objects for the clients to download; a mechanism for managing update, insert, and delete transactions; a method for handling queries to more than one database and combining the answers; and, a method for using the edited results of one query as part of another query. Although these missing elements require some challenging technical work, the individual elements already exist in one form or another. It should not take too long to integrate them into a specific system.

Whether or not a generic agent program can serve many database systems remains an open question. We are investigating several possibilities, but we also recognize the advantages of having a form or other front-end of a query builder tailored to a specific database system. Different groups of users have very different views of data collections. When users have an intuitive understanding of the method the agent program uses to elicit data requests, the learning curve rises and the need for extra documentation decreases. The agent provides a pragmatic method to help the user express data requirements in a familiar language.



Advantages of Clients <=> Agent <=> Servers Configurations

Clients <=> Agent <=> Servers configurations offer a number of technical and economic benefits to users. These benefits extend from better use of equipment to better distribution of computing tasks.

With an object playing the role of the Agent, the database administrators need to offer only a small number of objects in support of users with MS DOS and Windows, Mac, Unix flavors, and other popular systems. The objects require relatively little computing power and storage space. The clients' machines are merely presenting forms to users and writing data requests for the servers. This means that a large base of potential clients already have all of the equipment and software that they need to translate their data requests into the Servers' languages, transmit the requests over a network, and receive the results.

Since the servers distribute the agent objects to users, it takes less effort to keep the clients' configurations current and in sync with the database system. The servers can determine whether a client has the right version of the agent and update it where needed.

In a Clients <=> Agent <=> Servers configuration, clients transfer small program files to the servers. The servers extract relatively small tables from large collections of data. The configuration puts the computing power in the Servers where it counts. Searches and number crunching that would swamp typical desktop machines become routine on a more suitable RISC processor with more memory and faster disk access times.

The Agent has the ability to establish network connections, configure the client's view of database elements, draw up-to-date information from the database, display visual cues, interact with the client,

transfer data requests to the server databases, and present the results to the user. Like search agents in e-mail and reservation systems, the Agent works in the background and uses its connections to networked databases to link clients into servers. The more generic agent program stores methods for exchanging information with other objects. It follows data exchange protocols much as a librarian would use the Dewey Decimal System to locate a catalogued book. The Agent ideally reduces the slope of the learning curve for new users and helps all users focus on content rather than the technical details of electronic data storage and retrieval.

SUMMARY

Computing languages and icons have become familiar metaphors for the patterns of electronic signals that we use to control computing systems; so too the Agent in a Clients<->Agent<->Servers configuration gives us a useful metaphor for the visual front-ends and the network protocols that connect clients to databases through networks. The Agent makes it easier for humans to communicate a data request to database engines, and it puts a request in a form that improves the precision and reliability of the Servers. A working example shows some of the elements of the Agent. Advantages of the Clients<->Agent<->Servers configuration include lower costs of setting up Clients, better use of the computing power of Servers, and faster and easier system updates.

REFERENCES

ODBC Application Programmer's Guide, Microsoft, Inc.

SAS is a registered trademark of SAS Institute, Inc. in the USA and other countries. ® indicates USA registration.

AUTHOR CONTACT INFORMATION

Sigurd W.Hermansen and Greg Cox
Westat, Inc.
1650 Research Boulevard
Rockville, Maryland 20850
Work: 301/251-4268
Fax: 301/294-2092
e-mail: HERMANS1@WESTAT.COM
COXG1@WESTAT.COM