4-1-1997

An interactive distributed architecture for geographical modeling

Greg A. Wade
Southern Illinois University

David A. Bennett
Southern Illinois University

Raja Sengupta
Southern Illinois University

Proceedings of Auto-Carto 13, Seattle, Washington, 1997, pp. 306-316.

AN INTERACTIVE DISTRIBUTED ARCHITECTURE FOR GEOGRAPHICAL MODELING

Greg A. Wade, Researcher
Department of Computer Science
Southern Illinois University at Carbondale
Carbondale, IL 62901-4511

David Bennett, Assistant Professor and Raja Sengupta, Graduate Student
Department of Geography
Southern Illinois University at Carbondale
Carbondale, IL 62901-4514

ABSTRACT

The creation and modification of geographically explicit models is often difficult and time consuming due to the complexity and scale of the real world systems they simulate. Efficiencies can be gained by sharing models, model components, and data. Recent advances in computing technologies provide new tools that support distributed databases and modelbases. In this paper a distributed, platform independent system for geographic data retrieval and spatial modeling is presented.

1.0 INTRODUCTION

The types of analytical models and tools needed to address spatial problems are often domain specific and not well suited to generic software packages such as geographic information systems (GIS). Furthermore, the creation and modification of geographically explicit models is often difficult and time consuming due to the complexity and scale of the real world systems they simulate. Spatial decision support systems (SDSS) are designed to overcome some of the limitations of current GIS technology (Densham, 1991). However, SDSS that attempt to support the integrated management of geographical data and models are rare. Frameworks for geographical model management have been proposed (e.g., Bennett, in press; Wesseling et al., 1996) but they fail to take advantage of the ever growing capabilities of distributed computing. The work presented here builds on earlier work to create a distributed, platform independent system for geographic data retrieval and spatial modeling. To illustrate the utility of such a system a prototype

distributed Java-based SDSS (DJS) has been developed.

This implementation of a DJS allows decision makers to search network accessible repositories of data using geographical and contextual queries via the Internet or a private intranet. Models and atomic model components may also be retrieved from remote servers. Facilities are provided for linking existing elements retrieved from remote servers with newly created model components to form dynamic systems capable of simulating real-world events. System performance is enhanced by providing for the distribution of the database and modelbase across a network of heterogeneous computers. This allows for the use of divide and conquer techniques to solve computationally intense modeling problems (e.g., some machines can provide traditional GIS facilities while others on the network handle the computational tasks associated with performing complex simulations).

The relatively new computer programming language Java and its extensions contain the functionality needed to implement this project. Java provides a development environment for Internet applications and possesses unique features that facilitate the creation of software designed to be executed in a distributed environment. These features include platform independence and the ability to dynamically load and bind compiled code over the network (Gosling and McGlinton, 1996). Bennett's representational framework for geographical systems was modified and extended to support distributed objects, and its key software elements were ported to Java. Communication protocols and metadata requirements were defined and implemented to support network navigation and cross-platform geographical queries.

2.0 ARCHITECTURE OVERVIEW

The DJS is composed of five components tightly coupled via network communication: object repositories, model repositories, data servers, compute servers, and the DJS console. Figure 1 graphically illustrates the system elements and their relationship to the DJS console. These components may be distributed

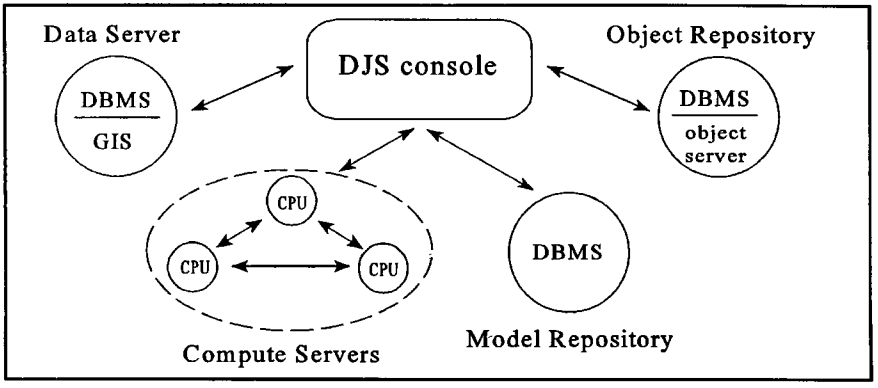

Figure 1. The DJS console's interaction with other system elements.

across heterogenous hosts linked via intranets or the Internet. The DJS console provides the graphical user interface to the system, and is the user's means of interacting with the other four components to create simulation models from model components.

Although each component is discussed as a disjoint entity, the design of the system does not dictate that each system element resides on a separate host. Nor is it required that each system element be on a single host. It is possible that a data server or an object repository may actually consist of several machines acting together as a virtual data server or object repository.

2.1 Object repositories

All data and models within this system are instantiated from Java classes. Class definitions are stored in object repositories that can be accessed across the network. These repositories consist of two elements, a database management system (DBMS) and an object server. The DBMS contains the names of Java classes, the hostname where specific classes are defined, and the TCP port number needed to access these object servers. These servers respond to requests by transferring binary representations of the required Java class over a network connection. This class can then be dynamically loaded into the DJS console or saved to disk for later use.

2.2 Model Repositories

Models are composed of a collection of model components. Each component represents an autonomous element of the simulation, and consists of static inputs, local variables, dynamic inputs, computations and outputs. Static inputs to a component include any value that does not change through the course of the simulation. Local variables and dynamic inputs are allowed to change during the simulation. Local variables are used by model components as state variables and get their values from the components calculations. Dynamic inputs on the other hand get their values from the outputs of other model components. Model repositories are used to locate model specifications or entire models stored in a DBMS. These specifications tell the system what inputs are required by the model or model component and what outputs or results are produced. In addition to locating these components on a model server, the user may create them interactively or load them from the local disk via the DJS console. The list of associated inputs is used to search for "data" needed to run the model. These data can be derived from existing geographic databases or from the output of other model components.

The flow of data from the outputs of one model component to the dynamic inputs of another occurs through *communication channels*. These channels utilize the communication technologies built into compute servers, thus allowing components running on different servers to communicate. The distribution of components can be used to exploit the natural parallelism of some models. Components not dependent upon each other for data can run concurrently as long as they have received all needed dynamic inputs. This form of parallelism follows

from the Dataflow computer architecture.

2.3 Data Servers

Data servers consist of two elements, a DBMS and a GIS. The DBMS is used to initially locate spatial data based on map extent and the data's attributes. In addition to the map extent and attributes for a data set, this database stores metadata required by the DJS. This information includes the hostname of the data server storing the data set, the filename the data is stored in on the server, the type of Java object the data may be loaded into, the individual who created the object definition, the creation date, and modification dates. Figure 2 shows an excerpt from the metadata of a soil's coverage for Jackson County, IL stored in a polygon data structure. After data is located the GIS is used to query and extract data residing on the remote server.

Description:	Soil id's from Jackson County Illinois
Host:	bast.cs.siu.edu
File:	/GeoData/Soil/IL/Jackson
Class:	dms.spatial.poly.PolyLayer
Projection:	UTM
ExtentMinX:	264,000
ExtentMinY:	4,160,000
ExtentMaxX:	311,100
ExtentMaxY:	4,204,100
NumAttributes:	1
Attributes:	Soil-Type

Figure 2. Sample metadata used by the DJS.

If data sets are located that require Java classes not installed locally on the user's computer, an object repository is contacted in an attempt to locate the compiled Java code for the class.

2.4 Compute servers

Compute servers are used to execute models consisting of components built to the user's specifications. Compute servers differ from the other system elements in several ways. They are capable of communicating among themselves while data servers, object repositories, and model repositories are only allowed to communicate with the DJS console. This is needed since model components must often exchange information. Secondly, they do not contain a DBMS. As their name implies compute servers are utilized solely for raw CPU power.

3.0 A SIMULATION EXAMPLE

To illustrate the modeling capabilities of the system, a simple stream network model was developed (Figure 3). This network consists of five segments, s_1 to s_5.

Each segment receives a constant inflow from overland runoff. Additionally segments s_3 and s_5 receive inflow from the outflows of segments s_1, s_2 and s_3, s_4 respectively. From these flows the outflow from segment s_5 is computed over time.

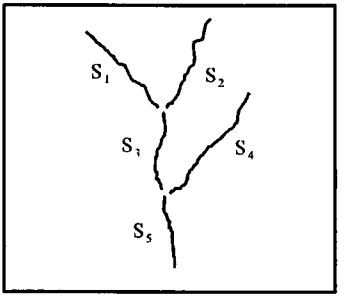

Figure 3. Stream network.

3.1 Simulating stream flow

Equations 1-4 summarize the Muskingum (Linsley et al., 1982) method of channel routing where I_1 and I_2 represent current and previous inflow respectively. Similarly O_1 and O_2 represent current and previous outflow respectively.

$$O_1 = c_0 I_1 + c_1 I_2 + c_2 O_2 \quad (1)$$

$$c_0 = (Kx - 0.5t) / (K - Kx - 0.5t) \quad (2)$$

$$c_1 = (Kx + 0.5t) / (K - Kx + 0.5t) \quad (3)$$

$$c_2 = (K - Kx - 0.5t) / (K - Kx + 0.5t) \quad (4)$$

The coefficients c_0, c_1 and c_2 (equations 2-4) are derived from K the storage constant (i.e., the ratio of storage to discharge), x the relative importance of inflow and outflow in determining storage and t the simulated time interval between computations. For most streams, x is between 0 and 0.3 with a mean value near 0.2 (Linsley et al., 1982). K can be approximated by the travel time through the reach.

3.2 Creating model components

A model component is created to represent each segment. Each component has three static inputs *(K, x,* and *over_land_flow)* which are initialized with values retrieved from the GIS, and an additional static input t supplied by the user. In addition to these inputs each component receives a dynamic input, *flow*, from the output of an upstream component. Any component not having a contribution from upstream flow, i.e., those representing segments s_1, s_2, and s_4, will always receive 0 for *flow*. Components also require the local variables in Table 1.

Variable	Initial Value	Purpose
c_0, c_1, c_2	0	coefficients for the Muskingum method
last_inflow	observed values loaded from the GIS	I_2 for the Muskingum method
last_outflow		O_2 for the Muskingum method
tmp	0	temporary variable

Table 1. Local variables used to simulate stream flow.

The component's variables are used in conjunction with the computations in Figure 4 to compute the components output, *flow*.

$c_0 = (Kx - 0.5t) / (K - Kx - 0.5t)$
$c_1 = (Kx + 0.5t) / (K - Kx + 0.5t)$
$c_2 = (K - Kx - 0.5t) / (K - Kx + 0.5t)$
flow = *flow* + *over_land_flow*
tmp = *flow*
flow = (c_0 * *flow*) + (c_1 * *last_inflow*) + (c_2 * *last_outflow*)
last_inflow = *tmp*
last_outflow = *flow*

Figure 4. Computations used to simulate stream flow.

This simulation could easily be extended by allowing the *over_land_flow* of each segment to vary over time. This value could be determined from the output of another model. The same approach could be taken to create inflows for s_1, s_2 and s_4.

4.0 IMPLEMENTATION DETAILS

As noted earlier the DJS was coded in Java. Development was done using Sun Microsystem's Java Development Kit (JDK) and Symantec's Café. This implementation utilizes four different programming techniques and tools.

4.1 Accessing the DBMS

The Java Database Connectivity (JDBC) API was used to access DBMS throughout the system. JDBC provides a convenient interface between the Java programming language and SQL databases. The JDBC API is implemented as a set of driver managers that can connect to various databases (Sun Microsystems, Inc., 1996). The JDBC driver chosen for the DJS implementation is a commercial product marketed by XDB Systems. XDB Systems' JetConnect JDBC driver in conjunction with their jetport server provide a bridge between JDBC and databases compliant with Microsoft's Open Database Connectivity (ODBC) standard (Ball et al., 1996).

Client software (i.e., the DJS console) using JDBC and XDB Systems' JDBC driver connect to the jetport server on a remote host, i.e., a machine acting as a data server, object repository, or model repository. The jetport daemon then accesses a Microsoft Access database via ODBC to perform database functions. See Figure 5.

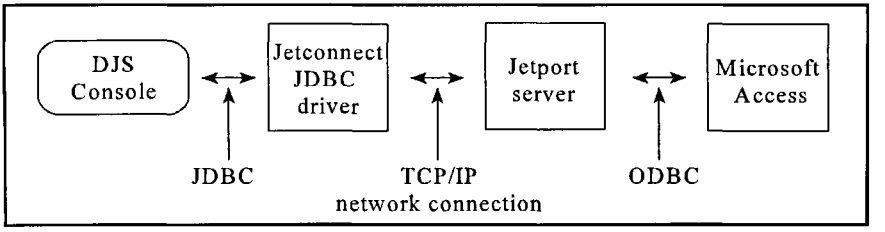

Figure 5. The DJS console's interaction with the data server's DBMS.

4.2 The object server

New models located by the search engine may require Java objects that are unknown to the computation host. As such, a mechanism must exist to export class definitions to compute servers and the DJS console. This is the role of the object server. To construct the object server a custom *ClassLoader* was created that is used to dynamically load compiled class files into the Java virtual machine. When Java classes, or objects instantiated from them, attempt to access an unknown class the class loader is invoked to locate the needed object code (Gosling et al., 1996). This allows the DJS console to load a class when a user attempts to access a data set requiring a spatial structure not available locally. The DJS console also implements a cache as part of its class loader to reduce the overhead of repetitively searching for and downloading classes.

4.3 Accessing the remote GIS

There does not currently exist an API similar to JDBC and ODBC for accessing remote GIS software. As a result, a radically different approach was used for accessing the GIS component of the data servers than was used for accessing the DBMS. However, the results are similar. In both cases the client sends a request, either an SQL or GIS query, and a result is returned. The GIS software is not accessed via an API. Rather, it is accessed by instantiating Java objects representing spatial data structures on the data server. A data file on the server is then loaded into the object. These remote objects are provided by HORB. HORB extends Java to provide object request broker capabilities that allow clients to create remote instances of objects and execute their methods on a remote machine across the network (Hirano, 1996). As such, the data is not transferred to the host running the DJS console. HORB accomplishes this through the use of proxy objects on the client. These proxy objects use the HORB library to communicate with a HORB server (Figure 6).

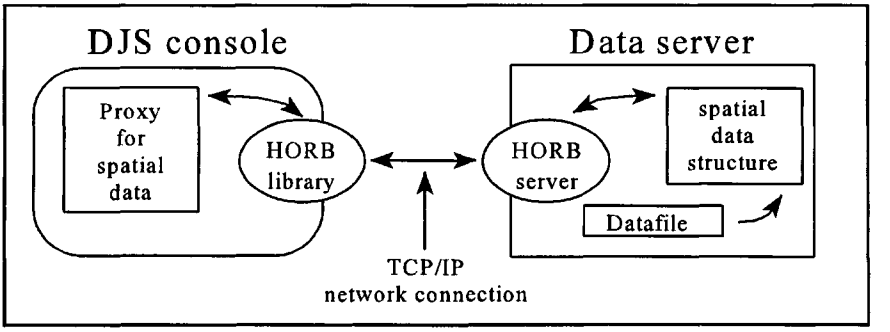

Figure 6. The DJS console's interaction with the GIS of a data server.

4.4 Creating model components and communication channels

The system also utilizes HORB to create model components on the compute servers. A model component object is instantiated on the compute server and methods are called to initialize its static inputs, local variables, dynamic inputs, computations and outputs. The component can then be started and stopped remotely. In addition to the model component, HORB is used to create a communication channel for each model component having outputs. This channel is created on the same compute server as the component sending outputs to the channel. Other components, again using HORB, then access this channel to retrieve dynamic inputs.

5.0 CURRENT LIMITATIONS AND FUTURE WORK

As with any prototype software system, this implementation of a DJS has several limitations. Most of these limitations all have solutions based in current geographic and computing technologies. Enhancing the current system through these technologies paves the way for future research.

5.1 The GIS

First the spatial data handling component of the data servers is primitive. Currently two types of spatial data structures, a vector and raster structure have been implemented. While these structures allowed for testing of the system, they provide only simple data retrieval. This limitation might be corrected by incorporating the work of Sengupta et al. (1996) which focuses on developing intelligent agents capable of interfacing with existing GIS products. This would allow the system to leverage the investment in pre-existing software and datasets. In implementing this approach, API's similar to JDBC and ODBC could be created for accessing GIS via these agents from Java.

5.2 Computational ability of model components

Currently the computational ability of model components is limited. Model components are only equipped to handle calculations consisting of basic

mathematical operations. However most real-world simulations require more advanced calculations. To resolve this problem a commercial package such as Mathematica or Maple could be used. The solution has one disadvantage: platform independence would be lost. An ideal solution would be to incorporate a Java-based symbolic mathematics library. Unfortunately, as of this writing, the authors know of no such package.

5.3 Increasing system performance

Lastly system performance needs to be improved. More of an effort needs to be done to exploit the parallelism of models. At present the DJS console simply cycles through its list of compute servers when creating model components, assigning a component to the next server in the list. The Vertically Layed (VL) allocation scheme to determine which model components to assign to which compute servers should be implemented. The VL algorithm uses heuristic rules in an attempt to maximize parallelism while minimizing communication overhead (Hurson et al., 1990). This would have the result of assigning model components capable of executing concurrently on different compute servers while assigning components which must execute sequentially to the same compute server. To even further enhance performance an optimization phase for the VL allocation scheme purposed by Kvas et al. (1994) could be added. This phase would take into account communication delays between compute servers. This would be essential for achieving high performance when utilizing compute servers spread across the Internet.

6.0 CONCLUSION

This implementation of a prototype DJS has illustrated the feasibility and functionality of a distributed platform independent geoprocessing SDSS. Current software technologies and techniques have matured to the point where creating such a system is possible. As a result a DJS can overcome some of the problems associated with current SDSS. In addition it can bring the power and flexibility of SDSS to many users who do not currently possess high-end computing resources. These users can run the DJS console on almost any personal computer or workstation supporting the Java virtual machine, and then access high performance computing facilities provided in house via an intranet or globally via the Internet.

ACKNOWLEDGMENTS

This work was funded in part by a grant from the Pontikes Center for the Management of Information, Southern Illinois University at Carbondale, Carbondale, IL 62901.

REFERENCES

Bennett, D.A. (in press). A Framework for the Integration of Geographic

Information Systems and Modelbase Management. *International Journal of Geographical Information Systems.*

Densham, P.J. (1991). Spatial decision support systems, in *Geographical Information Systems: Principles and Application,* ed. Maguire, D.J., Goodchild, M.F., and Rhind, D.W., Longman, London, pp. 403-412.

Gosling, J. and McGlinton H. (1996). *The Java Language Environment: A White Paper.* Sun Microsystems Inc.

Gosling, J., Yellin, F., and The Java Team. (1996). *The Java Application Programming Interface, Volume 1.* Addison-Wesley, pp. 19-22.

Hirano, S. (1996). What's HORB?. *http://ring.etl.go.jp/openlab/horb/doc/what.htm.*

Hurson, A.R., Lee, B., Shirazi, R., and Wang, M. (1990). A Program Allocation Scheme for Data Flow Computers. *Proceedings of the 1990 International Conference on Parallel Processing,* pp. I-415-I-423.

Ball, K., McClain, D., and Minium, D. (1996). *Bringing a New Dimension to Java Through Easy Access to Enterprise Data.* XDB Systems, Inc.

Kvas, A., Ojsteršek, M., and Žumer, V. (1994). Evaluation of Static Program Allocation Schemes for Macro Data-Flow Computer. *Proceedings of the 20th EUROMICRO Conference,* IEEE Computer Society Press, pp. 573-580.

Linsley, Jr., R.K., Kohler, M.A., and Paulhus, J.L.H. (1982). *Hydrology for Engineers.* McGraw-Hill, pp. 275-277.

Sengupta, R. R., Bennett, D.A., and Wade, G.A. (1996). Agent Mediated Links Between GIS and Spatial Modeling Software Using a Model Definition Language. *GIS/LIS '96 Annual Conference and Exposition Proceedings,* pp. 295-309.

Sun Microsystems, Inc. (1996). *JDBC Version 1.1 Release Notes.*

Wesseling, C.G, Karssenberg, D., Burrough, P.A., van Deursen, W.P., 1996, Integrating dynamic environmental models in GIS: The development of a dynamic modelling language. *Transactions in GIS,* 1(1):40-48.