A Method for the Integration of Existing GIS and Modelling Systems.

Paul Yates and Ian Bishop
Department of Geomatics,
University of Melbourne,
Parkville, Victoria, Australia, 3052.
pmy@sli.unimelb.edu.au
I.Bishop@engineering.unimelb.edu.au

Abstract

The integration of existing modelling and geographic information systems (including GIS) is an important activity that enhances the value of these systems. In this paper, we present a simple and comprehensive approach for the integration of separately developed software systems. Any information system can be integrated using our methodology without the complexities introduced by providing an interpretation of a universal language. The design of our integration methodology consists of four separate components, the protocol for communication, a message queuing system, wrapping software and an integration manager. Relevant conceptual models and implementation techniques are discussed in the paper. We also describe some examples of the software we have successfully integrated and present an example script for managing a simple integration activity.

1 Introduction

Currently, there are many spatial database management systems (GIS), aspatial database management systems and modelling systems used in modelling activities. In general, these software systems are, and have been, developed independently with their own specifications, interfaces, data models and data types. As it is often the case that information and operations needed for a particular task exist in separate information systems, there is a need to integrate these information systems. We note that there are many different meanings given to the term integration. These range from providing a simple methodology to allow the cohesive operation of the separate information systems, through to providing a high level language and data model that encompasses all the operations and complexities of each separate information system (e.g., the "Universal GIS Operators" described by Albrecht (1995) and the Open Geodata Model of the Open GIS Consortium (1996)).

An example of the need for integration is found in the urban modelling area. Wegener (1994) surveys the state of the art in operational urban models. In the survey several different urban subsystems are identified and several urban modelling systems are identified that model some or all of the urban subsystems. There is a strong interest in being able to integrate these models with GIS. Also, where models do not cover all urban subsystems, there is a need to be able to integrate the models so as to increase the number of subsystems covered.

In this paper, we present a simple and comprehensive approach for the integration of separately developed software systems. Any information system can be integrated using our methodology without the complexities introduced by providing an interpretation of a universal language.

An example of such complexity is the need to translate the universal language into a language understood by an individual information system. In the research area of federated information systems based on relational and object oriented databases the translation process is less difficult

as there already exist standard languages such as SQL and OQL understood by most, if not all, databases that need to be integrated. Given this fact, during the integration process the developer can concentrate on other problems related more specifically to the data and data structure (such as schema translation and schema integration (Sheth & Larson 1990)). When integrating software supporting geographic information systems and modelling there is seldom a common language to the systems being integrated. This means that the developer must be concerned with the language, data model and data structures of each system. This is a primary difference from the standard requirements for the integration of relational or object oriented database systems.

The OGIS guide (Open GIS Consortium 1996) identifies several software layers in the design of integration software. These include, the presentation layer, the application and application server layer, the spatial data access provider layer, the database layer and the hardware and network layer. Our proposed methodology concentrates on the implementation of the application and database layer. We do not directly address the important issues relating to the development of a high level language and data model of the spatial data access provider layer. It is significant to note that to be able to integrate information systems at a high level, there is also a need to integrate to the level described in this paper. What we directly address in this paper are issues relating to the design and implementation of a system that allows concurrent access to data and programs (in their current form) provided in disparate information systems. Most importantly, we use languages native to each of the individual information systems to access the data and programs.

There are many important issues to consider in the development of a system to integrate disparate information systems. These issues include, data access, interoperability, integration process management, user interface design and security. Data access encompasses the requirements for transformation between data models and translation between data types as well as the communication of the data between software systems. By interoperability we mean

the ability to access the modelling systems (programs) available in different information systems. It is important that there exists the ability to control the integration process, i.e., there is a need to provide for the specification of the steps required to perform a given task needing integration of separate software systems. Currently, we are not immediately concerned with the provision of a user interface; at present we provide an interpreter for a small language to manage the integration process. Eventually, we intend to provide a "drag-and-drop" type interface linking models to data sets. However, to provide such an interface we will need to address issues related to providing a universal language for integration. Another issue that is beyond the scope of our current implementation is security. This includes concepts relating to the rights to use the data and software, and auditing of such use.

2 Background

2.1 Conceptual Models

Integration of existing software systems has been the subject of much recent research. In particular, Abel, Taylor & Kuo (1997) develop a theory for the integration of modelling systems for environmental management information systems. In the model several generic concepts relating to software integration are identified. The concepts of an object, problem, solved problem, solver, execution plan and a well-defined problem are all defined. A significant point is that they equate the concept of a problem with the concept of a query in a database system. A solver then provides a solution to a problem in a similar fashion to the way a database provides an answer to a query. Their model provides a conceptual framework that enables both a proper description of a given integration activity as well as a description of the software components used to develop a solution for an integration problem.

Wiederhold (1992) develops the concept of mediators for information systems. The paper discusses an architecture for an information system consisting of three layers, a user layer, a mediator layer and a base layer (possibly consisting of multiple databases). The mediator layer of an information system sits between a user layer and a base layer. It is

the responsibility of the mediator layer to accept requests, distribute the requests to the appropriate information system in the base layer and collate and return the result of a request to the user layer. The mediator layer may make use of knowledge about the request (and the data required to answer the request) to decide how to distribute the request to the base layer. Buneman, Raschid & Ullman (1997) propose a "mediator language" in which it is possible to describe the data structures and data models that are part of a given information system. In addition, the language allows for the expression of database queries which can then be passed to a given information system.

Making an independently developed software system communicate often requires the system to be "wrapped", i.e., a piece of software is developed that communicates to external processes as well as controlling the systems being integrated. The use of wrappers is a common component in a number of software architectures used for integrating software (Buneman et al. 1997, Ishikawa, Furudate & Uemura 1997, Papakonstantinou, Gupta, Garcia-Molina & Ullman 1995, Wiederhold 1992). The wrapping software provides a shell around the software to be integrated, providing a point of access to the integrated software.

2.2 Implementation

Implementation of a system for the integrating software inherently makes use of pre-existing approaches that support the development of distributed systems software. Examples of such approaches include the Remote Procedure Call (RPC) (Comer & Stevens 1993), the message queuing model (Blakeley, Harris & Lewis 1995).

In the remote procedure call paradigm, calls to procedures that do not exist in the calling program are passed to a remote program for execution. The calling program (or client program) then waits for the completion of the called procedure before continuing execution (exactly as it would if the procedure was local). Data is passed to and from the remote program using a common data representation (such as the External Data Representation XDR). Such a method of communication is called synchronous, as the calling program waits until the called procedure returns before continuing. In a message queuing system there is a message queue associated with each participating system. Using message queuing, a system communicates by placing messages on a queue associated with the system with which it needs to communicate. The called system will then retrieve the message from its queue when it is ready. The calling system is free to continue processing or wait depending on its own needs. Hence, the communication can be synchronous (as with the RPC mechanism) or asynchronous. Commonly, the calling program only directly communicates with a program, called the queue manager, whose specific duty is to manage the queues associated with each program.

Blakeley (Blakeley et al. 1995) defines a criteria for the selection of the appropriate style of communication. Significantly, the use of message queuing systems is most appropriate when there is a mixture of application types, old and new programs and network types and where the programs are highly independent. These criteria are common with the requirements for integrating GIS and modelling systems.

Once a message queuing communication process has been established, each integrated process must be capable of understanding the message it is passed. The usual method of passing understandable messages is to define a protocol. The definition of a protocol defines the structure and interpretation of the messages that can be passed amongst the integrated systems. For example, suppose we have a protocol containing a command "exec" with one string parameter that asks for the execution of the string it is passed. A message "exec union covera coverb coverab" passed to an ARC/INFO process would instruct ARC/INFO to execute a union operation between the coverages a and b.

3 Architecture

Several conceptual models and methods for implementation were identified in the previous section. Issues that we considered important while designing and implementing the integration software are:

(1) the integration of new software components should require the minimum amount of programming effort;

(2) the protocol for communication should contain the smallest set number of commands necessary to enable integration (i.e., the smallest set enabling one software component to make calls to functions provided in another software); and

(3) there should exist some ability to control the integration process through the use of an integration language.
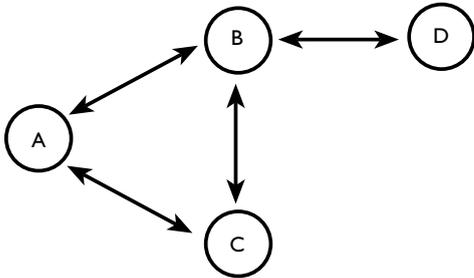

Figure 1: Asynchronous messaging between process.

3.1 Components

The design of our integration methodology consists of four separate components, the protocol for communication, a message queuing system, wrapping software and an integration manager. We begin with the definition of a protocol for which the separate systems communicate. To meet the minimum requirements, we have developed a protocol that includes methods to: establish and close communication; and execute programs and/or scripts. The parameters of the establish communication command include the specific location queue manager and the location of the process that is making itself available for integration. The contents of the execute command is text understood by the integrated process. To accept and request data we make use of the existing file transfer protocol definition.

The second component consists of a message queuing system. The message queuing system manages a queue for each path of communication that is established. Messages are passed to the message queuing system from an integrated system and placed on the appropriate queue. They then remain on the queue until they are retrieved by the

appropriate system. Hence, the communication between integrated systems is asynchronous. It is important to note that we specify that the communication between a given system to be integrated and the message queuing system is synchronous. Figure 1 shows an example of the communication between four systems. The communication at this level is asynchronous. Figure 2 shows the actual communication paths that exist for the abstract communication paths shown in figure 1. Messages from A to B are placed on B's queue. Messages from B to A are placed on A's queue.

The third component consists of software to implement the wrapping of the information systems to integrate. At the moment this software is set up to communicate with the queue manager. We do not specify the method of communication between the wrapping software and the system to integrate as this depends upon the system being integrated.
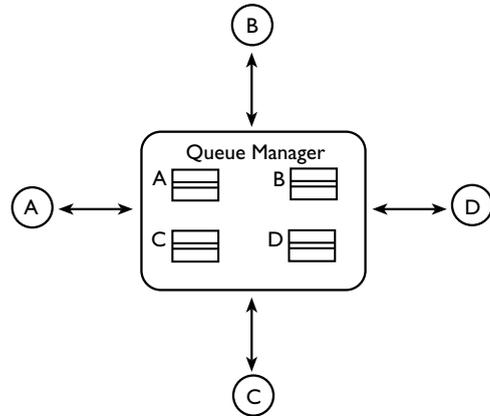

Figure 2: Synchronous messaging between processes and the queue manager.

For example, ArcView has the ability to communicate using the RPC mechanism, other software may not have this ability or there may be some other preferred method of communication. The wrapping software does not interpret the string passed for execution. That string is passed to the wrapped software for interpretation.

The fourth component of our design consists of the specification of a small interpreted language to manage the integration process. The basic elements of the language are

the commands, the open, exec, send and mode. The use of the open command allows for the definition of multiple connections to different software systems. Messages can be sent to defined connections through the exec command. Such messages sent through the connection are specific commands relevant to the particular software system on the connection, hence, the use of "exec" for the name of the command. For example, for the connection to an ARC/INFO session a message may contain a particular ARC/INFO command or macro. As several open connections to different software systems can be defined, a single script written in the integration language can contain a mix of the languages available to different software systems. The mode commands provides a method to specify the type of communication (synchronous or asynchronous). This command is useful for the cases in which the processing of a given script must be done synchronously. Finally, the language also includes commands to send data to a connection and request data from a connection. Figure 3 shows a simple script for running an urban model on data

```
mode.sync      # Set to synchronous mode of communication mode.sync
# Declare connection to arc/info on machine scamper and
# an urban modelling package on machine daisy and buttercup
open arc scamper, um1 daisy, um2 buttercup
# Export data in preparation for use in urban model
arc.exec workspace /usr9/pmy/urban/in ; gridascii house house.asc ;
gridascii pop pop.asc ; gridascii emp emp.asc
# Send data from scamper to daisy and buttercup
send scamper:/usr9/pmy/urban/in/*.asc daisy:/usr4/people/pmy/umdata/in
send scamper:/usr9/pmy/urban/in/*.asc buttercup:/home/pmy/umdata/in
# Mode can now be asynchronous for the execution of models
mode.async
# Step the urban model by one step ... input parameter 0.05
um1.exec step 1 0.05 house pop emp
# Step the urban model by one step ... input parameter 0.95
um2.exec step 1 0.95 house pop emp
# Reset mode to synchronous (ie wait for models to finish)
mode.sync
# Return stepped data
send daisy:/usr4/people/pmy/umdata/out/*.asc scamper:/usr9/pmy/urban/out1
send buttercup:/home/pmy/umdata/out/*.asc scamper:/usr9/pmy/urban/out2
# Re-import data
arc.exec workspace ../out1 ; asciigrid house.asc house ;
asciigrid pop.asc pop ; asciigrid emp.asc emp
arc.exec workspace ../out2 ; asciigrid house.asc house ;
asciigrid pop.asc pop ; asciigrid emp.asc emp
# Compute the difference in the population values and gridshade it
arc.exec workspace ../out ; grid ;  popdiff = ../out1/population - ../out2/population " quit
# Close connections
close um,arc
```

Figure 3: A simple example script for integrating ARC/INFO with an urban modelling package

stored in ARC/INFO. The model is run twice with a different input parameter on different machines. Note the two executions of the model are done simultaneously on different machines.

3.2 Implementation Process

Given the components described in the previous section, the process of integration consists of two steps. First, the wrappers understanding the above protocol are developed for each the information systems. Wrappers can be implemented in a language such as C or using a portable scripting language such as TCL while making use of tools such as Expect for automating interactive applications (Libes 1994). The second step is to write a script for controlling the integration process. This script is interpreted using the integration manager.

Use of the integration manager is not always mandatory. For example, it may be the case that the software being integrated can communicate with the queue manager without the need to use the integration manager. For example, ArcView includes RPC classes and the wrapper understanding the integration protocol can be built using Avenue. Other Avenue scripts may also also place messages on the queue of another process independently of the integration manger. In the case where there does not exist some communication ability within the software being integrated, it may be necessary to drive the process using the integration manager. Consider integrating several executable programs that cannot themselves place messages on a queue. Such examples can be found in urban modelling where different previously developed software may be concerned with modelling different urban subsystems (such as transport and employment). The protocol we use does not have the ability to initiate and control a series of steps executed by separate software systems. To do so, we have introduced the integration manager language. Scripts written in the integration manager language provide the necessary steps to perform a given task.

3.3 Examples

Currently, we have implemented such software to wrap ARC/INFO so as to enable it to be integrated with modelling software written in the unix environment. ARC/INFO, through its inter-application communication (IAC) provides RPC connection to other processes. The wrapper simply interprets the messages passed to it in the following way: a request for a connection starts ARC/INFO in a server mode; an execute command passes the string to the ARC/INFO session; a close connection request closes the ARC/INFO session. Some other software systems that we are looking to integrate includes GENAMAP and Illustra (an object-relational database management system).

Another example can be found in our integration of ArcView (running on a Sun) and a piece of visualisation software developed using the Performer toolkit (on an SGI). This integration only makes use of the queue management software. Both ArcView and the Performer based program act as clients to the queue manager. The integration of these programs was done for a specific project in which a polygon that is selected using ArcView is highlighted in a three-dimensional scene viewed using the Performer toolkit.

4 Summary and Future Directions

In this paper we presented a method to integrate existing information systems. We did not require the implementation of a high level language for integration, but took a more simplistic view of integration concentrating on minimum requirements necessary to enable communication and sharing of procedures between systems. In our design and implementation we have concentrated on issues relating to interoperability. The method is comprehensive in the sense that most (if not all) software can be wrapped with software understanding the protocol we have defined.

Other work on software integration for GIS and modelling systems has concentrated on the definition of a high level language, data models and data structures for integration. We have not addressed specific issues relating to such languages. However, note that the high level languages would need to be translated into calls on the individual systems. This may be possible through the translation of the language into the integration management language

described in this paper. This translated script could then be interpreted using the software describe here.

Although, we have currently implemented both the interpreter for the wrappers and management language in C, there is no reason that we cannot use another language to implement these systems. In fact, we intend to implement the interpreter as a Java applet enabling its use through any system containing a Java interpreter (e.g. Netscape).

References

Abel, D. J., Taylor, K. & Kuo, D. (1997), 'Integrating modelling systems for environmental management information systems', SIGMOD Record 26(1), 5—10.

Albrecht, J. (1995), Universal Analytical GIS Operations: a task-oriented systematisation of data structure-independent GIS functionality leading towards a geographic modelling language., PhD thesis, University of Vechta, Germany.

Blakeley, B., Harris, H. & Lewis, R. (1995), Messaging and queuing using the MQI : concepts & analysis, design & development, McGraw & Hill Book Company Inc., New York.

Buneman, P., Raschid, L. & Ullman, J. (1997), 'Mediator languages — a proposal for a standard', SIGMOD Record.

Comer, D. E. & Stevens, D. L. (1993), Internetworking with TCP/IP Vol III: Client-server programming and applications, Prentice-Hall Inc., New Jersey.

Ishikawa, Y., Furudate, T. & Uemura, S. (1997), A wrapping architecture for IR systems to mediate external structured document sources, in R. Topor & K. Tanaka, eds, 'Database systems for advanced applications '97', Vol. 6 of Advanced Database Research and Development Series, World Scientific Publishing Co., Singapore, pp. 431—440.

Libes, D. (1994), Exploring Expect, O'Rielly & Associates Inc.

Open GIS Consortium (1996), The OpenGIS Guide, Introduction to Interoperable Geoprocessing, Part I., http://www.ogis.org/guide/guide1.htm.

Papakonstantinou, Y., Gupta, A., Garcia-Molina, H. & Ullman, J. (1995), 'A query translation scheme for rapid implementation of wrappers', Lecture Notes in Computer Science 1013, 161—186.

Sheth, A. & Larson, J. (1990), 'Federated database systems for managing distributed, heterogeneous and autonomous databases', ACM Computing Surveys 22(3), 258—274.

Wegener, M. (1994), 'Operational urban models, state of the art', Journal of the American Planning Association 60(1), 17—27.

Wiederhold, G. (1992), 'Mediators in the architecture of furture information systems', IEEE Computer pp. 38—49.