Using DCOM To Support Interoperability In Forest Ecosystem Management Decision Support Systems

W.D. Potter, S. Liu, X. Deng
Artificial Intelligence Center, GSRC 111, University of Georgia, Athens, GA, 30602
H.M. Rauscher
USDA Forest Service, Bent Creek Experimental Forest, Asheville, NC
(Contact W.D. Potter at potter@cs.uga.edu)

Abstract

Forest ecosystems exhibit complex dynamics over time and space. Management of forest ecosystems involves the need to forecast future states of complex systems that are often undergoing structural changes. This in turn requires integration of quantitative science and engineering components with socio-political, regulatory, and economic considerations. The amount of data, information and knowledge involved in the management process is often overwhelming. Integrated decision support systems may help managers make consistently good decisions concerning forest ecosystem management. Integrating computer systems using a system-specific or custom approach has many disadvantages. We compare a variety of current approaches, suggest characteristics that an approach should have, and propose that DCOM (Distributed Component Object Model) is an approach that is very suitable for forest ecosystem decision support system integration.

Keywords: Interoperability, Decision Support Systems, DCOM, System Integration

1. Introduction

Decision Support Systems (DSSs) may offer help in making decisions in situations where the problem-solving process requires a key contribution from expert judgment but human information-processing limitations impede decision making (Rauscher 1995). The ultimate objective of a DSS is to assist decision-makers (e.g., managers, planners, public officials, scientists, and the general public) in planning and the decision-making processes by giving the decision-makers useful and scientifically sound information.

A DSS may consist of a number of subsystems, each with a specific task. In forest ecosystem management (FEM), a DSS may contain a user interface, database, geographical information system (GIS), knowledge base, simulation and optimization models, help/hypertext management, and data visualization and decision methods (Rauscher 1999).

Different forest ecosystem management decision support systems (FEM-DSSs) support different parts of the ecosystem management process (Rauscher 1999). FEM-DSSs may be categorized as either "full service" or "functional service" systems. Full service FEM-DSSs attempt to be comprehensive by offering support for the complete forest ecosystem management process. In addition, these full service systems (also called full service modules) may be classified by their specific level or scale of support: regional assessment, forest planning and project-level planning. Functional service FEM-DSSs provide more narrowly specialized support for one or a few phases of the forest ecosystem management process. These service modules are further categorized by their function, for example, group negotiations, vegetation dynamics, disturbance simulation, or spatial visualization.

Mowrer et al. (1997) reviewed over thirty FEM-DSSs and reported several noteworthy conclusions. First, they found no single system that successfully addresses every important aspect of forest ecosystem management. Second, none of the systems comprehensively address ecological and management interactions across multiple scales. Third, the current generation of FEM-DSSs is much less capable of addressing social and economic issues than biophysical issues. Finally, no system simultaneously considers social, economic and biophysical issues[1] and only one system provides group consensus-building support.

Most FEM-DSSs were developed independently of one another. As a result, they are typically large, monolithic, stand-alone systems incapable of performing joint-problem solving tasks without extensive revisions. Because none of the existing FEM-DSSs have been found capable of addressing the full range of support required for the management of a complex forest ecosystem (Mowrer et al. 1997, Rauscher 1999), an ideal FEM-DSS requires the combined capabilities of many of the available systems to work together. This necessitates both full service and functional service systems integration. Furthermore, it is often more cost effective to re-use existing software than to develop custom software when an existing FEM-DSS is to be enhanced to provide additional services.

FEM-DSSs and/or their component subsystems have been written in different software languages, reside on different hardware platforms, have different data access mechanisms, and

---

[1] Although EMDS falls into the category of systems that do not do these things, it is capable of supporting these if the user provides the proper design and implementation (Reynolds et al. 1997).

different subsystem interfaces. For example, non-geographical databases may be written in Oracle, geographical information system (GIS) databases in ARC/INFO, knowledge bases in Prolog, and a simulation model in Fortran. This kind of development heterogeneity has to be addressed when integrating FEM-DSSs. Although efforts at integrating FEM-DSSs do exist, they have been *ad hoc* yielding unique, point-to-point custom solutions. These custom solutions are typically difficult to maintain and extend (Rauscher 1999).

One solution to many of the problems in systems integration is to provide an interoperable architecture for software systems (Potter et al. 1992, Potter et al. 1994, Otte et al. 1996). Interoperability is the ability of two or more software components to cooperate by exchanging services and data with one another, despite the possible heterogeneity in their language, interface and hardware platform (Heiler 1995, Wegner 1996, Sheth 1998). Interoperable software architectures provide a standard that promotes communication between components, and provides for the integration of legacy and newly developed components.

In the past decade, organizations have been moving mainframe-based systems toward open, distributed computing environments. A distributed computing environment is an environment where multiple computers are networked together, and allowed to share data and processing responsibilities. The demand for interoperability has been driven by the accelerated construction of large-scale distributed systems for operational use and by increasing use of the Internet (Manola 1995). Distributed computing offers many advantages, including location transparency to users, scalability (adding more capability by adding more computers to the network), fault tolerance (allowing processing to continue even when a computer or network connection is broken), load balancing (sharing the work load equally among the networked computers) and resource sharing (sharing databases or other special services). As such, much discussion of interoperability and its research have been concerned with distributed computing; for example, the recent emergence of Java, the Object Management Group's CORBA (Common Object Request Broker Architecture) and Microsoft's DCOM (Distributed Component Object Model) are all for this purpose. In addition, object orientation (OO) is probably the most widely used approach in software development and the basis for the CORBA and DCOM interoperability architectures. OO is an organization scheme for sets of objects that interact, and share data and operations. OO makes it easier to maintain software modules, and makes it possible to re-use

existing software objects. Consequently, platform independence, as well as language independence, has been a major focus in these interoperability architectures.

Designing, implementing, and maintaining interoperable software architectures for FEM-DSSs are urgent and challenging tasks for current FEM-DSS developers. But, which architectural approach is best suited for FEM-DSSs? This paper examines several existing approaches to software interoperability. First, we present an overview of the different types of interoperability. Then we analyze the interoperability architectures of five representative FEM-DSSs followed by an examination of four computer science approaches. The remaining sections identify the design criteria for an effective interoperability software architecture and describe a prototype developed using DCOM. In addition to existing interoperability approaches, we take a brief look at a new approach that may have a significant impact on distributed systems. This approach is based on issuing commands to distributed software objects via the Internet (Walsh 1998, Winer 1998, Udell 1999).

2. Interoperability

The scope of interoperability has changed drastically since the first introduction of a few interconnected computers. Currently we think of the scope as including any application on any computer interfacing with any other application on any other computer; a global scope (Sheth 1998). However, having a global perspective introduces a variety of issues that need to be addressed in order to understand what interoperability really is. The following levels or types of interoperability have been identified:

- Platform interoperability (also known as location and technical interoperability) resolves the differences in the hardware, system software, and the services that deal with communication between two objects (Lockemann et al. 1997, NC3A 1997). It allows a client to make a transparent call to a server even if the server runs in another process or on another computer, so that the call looks as if it were an in-process (local) call.
- Basic interoperability means that binary components (executables) uniquely developed by certain developers are assured to function with other binary components built by different developers.

- Versioning interoperability is the agreement that one system component can be upgraded without requiring all other system components to be upgraded.

- Language interoperability provides language independence such that applications that are implemented in different programming languages can be integrated (Meek 1994). As a result, components written in different programming languages can communicate with each other.

- Notational interoperability (Lockemann et al. 1997) can be further classified into data interoperability, object interface interoperability, and object framework interoperability. Data interoperability addresses disagreements on data formats, types, structures and representations (Manola 1995, Wegner 1996). Object interface interoperability deals with agreements on object interface characteristics (Manola 1995). Object framework interoperability is concerned with cooperation among the sets of object classes found in object frameworks (Gamma et al. 1995). Incompatible or different functional interfaces, data models, data types, database schemas, terminology, and data formats are example issues to be addressed by notational interoperability (Lockemann et al. 1997).

- Semantic interoperability ensures that exchanges of services and data make sense, that is, that the client and the server have a common understanding of the meanings of the requested services and data (Heiler 1995). Semantic interoperability is often closely linked to notational interoperability because issues associated with notational interoperability can and do lead to semantic conflicts in many cases (Heiler 1995).

- Coordinational interoperability deals with the interaction between client and server operations, for example, preservation of temporal as well as functional properties (order constraints on operations or coordination of inputs from multiple input streams) (Wegner 1996). Coordinational interoperability is usually obtained through common policies, contracts and protocols to which the components subject their activities (Lockemann et al. 1997).

It is technically more difficult to achieve semantic interoperability and coordinational interoperability (both considered to be high-level types) than platform interoperability, language interoperability and notational interoperability (all considered to be low-level types). For example, interoperability can be realized with reasonable efforts for a wide range of differences in data formats and for recognized differences of representation (Wegner 1996). Semantic interoperability, however, may be hard to achieve in many cases (Heiler 1995). Semantic

agreement is often lacking when old data or procedures are used for new purposes not anticipated by their original developers. Semantic agreements may also be lacking among new systems that are the products of independent development efforts (e.g., systems developed at different times by different programmers). Finally, determining the necessary semantic information with accuracy can be very difficult if not impossible (see Heiler 1995 for further explanation).

There are two major mechanisms for interoperation: *interface standardization* and *interface bridging* (Wegner 1996). Interface standardization links client and server interfaces to predefined standards whereas interface bridging is a two-way linkage between a client and server. Interface standardization needs only $m + n$ links for m clients and n servers, as opposed to $m * n$ links for interface bridging. Therefore, interface standardization is more scalable and reduces the task of interconnecting components, but standardized interface systems are closed and thus may preclude supporting new functional features like transactions that are desired later but not considered at the time of standardization. Interface bridging, on the other hand, is open and more flexible for tailoring requirements of particular clients and servers, yet it carries a high price in custom development, maintenance, and lack of extensibility.

3. FEM-DSS Interoperability

In order to investigate interoperability among FEM-DSSs, we analyzed five representative systems selected from Mowrer et al. (1997). The systems selected were NED-1 (Twery et al. 1997, Rauscher et al. 1997), LOKI (Bevins and Andrews 1993, Bevins et al. 1995, Keane et al. 1996), FVS (Teck et al. 1996, Teck et al. 1997), LMS (McCarter et al. 1998, McCarter et al. 1999), and INFORMS (STARR Lab 1993, Perisho et al. 1995, Choo and Lee 1997). NED-1, LOKI, LMS, and INFORMS are full service FEM-DSSs. FVS, in contrast, is a functional service system specializing in forecasting general vegetation dynamics. However, FVS is highly modularized with an intricate, ad-hoc inter-module approach to communication that was interesting enough to warrant examination. Each FEM-DSS was subjectively evaluated against the following seven criteria: language independence (i.e., language interoperability), platform independence (i.e., platform interoperability), architectural level, module interaction, legacy handling, object orientation and distributed processing capability.

A well-designed interoperability architecture for module interaction should be generic and general-purpose. It should have a well-defined standard for communication-related issues such as object registration (a scheme to identify available objects and their function), discovery (finding a new component when it becomes available) and cross-network transport protocols (the standard followed by different networks that are connected such as the Internet with a corporate local area network). If the mechanism is an *ad hoc*, point-to-point solution that has, for example, domain constraints, it will be of little use to others.

Integration of software modules involves dealing with legacy components in many cases. These legacy components often possess a user interface that is difficult to manage in an integrated environment. This is due to the fact that the user interface is typically deeply intertwined with the functioning of the legacy application.

A legacy system with an OO framework that allows object reuse may facilitate the integration of that system. Because the Internet offers a new way of delivering knowledge to the public, one would be better off if the system supports it. Furthermore, a mature, well-documented standard with a large customer base is usually favored over a system that is still in the conceptual or prototype stage. Finally, the cost of the architecture has to be factored in. Given that other factors are comparable, an interoperable framework that is inexpensive or even free may be more competitive.

Table 1 summarizes our comparison of NED-1, LOKI, FVS, INFORMS, and LMS (see Liu 1998 for complete details). We arrived at several findings during this evaluation process. First, these systems integrate a diverse range of knowledge information (e.g., decision making, knowledge base, simulation modeling, GIS, associated databases for calibration and execution, data visualization, etc). Independent construction, and a lack of USDA Forest Service agency-wide data bases, data standards, and standard hardware, have led to each of these systems being developed with their own localized data requirements (data formats, data base and GUI etc). There was a lack of documentation for most systems regarding their architectural designs; only system functionality was typically well described. Except for LOKI, the interoperable frameworks for all systems evaluated are at a high level, meaning that they are all domain-specific designs, resulting in unique, point-to-point solutions to module communications that are difficult to generalize into generic architectures. LOKI is a promising middleware architecture that deserves

a follow-up investigation and evaluation for its potential agency-wide use, but its modified implementation has yet to be distributed and its generic features remain to be seen. None of the FEM-DSSs evaluated supports language interoperability, platform independence, distributed processing and the Internet. Most systems (except for NED-1 and possibly LOKI) are not object-based, therefore having no object reusability. Finally, none of the FEM-DSSs we examined defines an explicit standard mechanism for dealing with the peculiarities of legacy applications. Although LMS was designed specifically to integrate existing systems, it does so in an *ad hoc*, point-to-point manner.

4. Non-FEM Approaches

Interoperability outside the forestry domain has received extensive attention. We evaluated four such approaches using the same criteria as before (see Liu 1998 for details). The approaches we addressed include CORBA (the Common Object Request Broker Architecture, OMG 1997, von Bultzingsloewen et al. 1996, Kramer et al. 1997, Leppinen et al. 1997), DCOM (the Distributed Component Object Model, Microsoft 1995, 1996, 1997), intelligent agent-based software engineering (Finin et al. 1994, Genesereth and Ketchpel 1994, Mayfield et al. 1996), and DIAS/DEEM (the Dynamic Information Architecture System/Dynamic Environmental Effects Model, Argonne National Laboratory 1995a, 1995b). These approaches encompass several different areas of computer science, including databases where the emphasis is on interoperation and data integration, software engineering where tool and environment integration issues dominate, artificial intelligence where systems consisting of distributed intelligent agents are being developed and explored, and information systems.

These four approaches are designed to provide various sorts of software interoperability and easy integration of legacy systems. Table 2 compares and contrasts the four approaches. CORBA and DCOM are two mature, interoperable object models that provide well-designed integration and communication standards. Both are considered low-level architectures. They are used as the backbones or middleware of modern distributed object environments. They facilitate the task of building distributed applications by presenting the network as one large virtual machine in which remote objects appear to be local. Middleware use frees the developer from many of the low-level programming tasks necessary to achieve integration of and coordinated interactions

between distributed objects (such as handling the individual data packets sent from one machine to another), while at the same time providing many necessary support services. Neither CORBA nor DCOM assume any domain constraints and can both be applied to the integration of any applications. They are generic, general-purpose integration frameworks that are very popular.

CORBA and DCOM both provide standard specifications for achieving language interoperability and platform independence. They define their own interface standards to deal with peculiarities of legacy applications. They support distributed processing, object reuse, and the Internet. Both architectures are well documented and the documentation materials are easily accessible to the public – on-line as well as through books and journal articles. CORBA can be purchased from multiple vendors, and DCOM is shipped with Windows NT/98 or can be downloaded free for Windows 95.

The intelligent agent-based approach and DIAS/DEEM are high-level frameworks in that, primarily, neither intends to address low-level details (e.g., object registration and discovery, cross-network transport) about interoperability. Instead, they use available technologies such as CORBA or DCOM as their respective facilitating middleware. The intelligent agent-based approach attempts to address high-level (such as semantic) interoperability, especially for large, complex systems whose components need to interact autonomously and intelligently in a heterogeneous, distributed environment. In the intelligent agent-based software engineering approach, the agents must possess a variety of abilities. They must be able to: communicate with each other using an expressive communication language; work together cooperatively to accomplish complex goals; act on their own initiative; and use local information and knowledge to manage local resources and handle requests from peer agents. It is a very promising methodology, but its implementation tends to be expensive because of its technical complexity (Finin et al. 1994, Genesereth and Ketchpel 1994) and it has not matured enough for wide use.

DIAS/DEEM specifically deals with integration of environmental simulation models and therefore is not a general-purpose architecture that can be easily applied to other domains. DIAS is an OO framework into which simulation models, information processing applications and databases can be integrated. DEEM is an example application of DIAS that provides interoperability among environmental models.

5. Design Criteria For An FEM-DSS Interoperability Architecture

As we have seen, several FEM approaches claim to support an interoperable architecture. However, there are many drawbacks to these approaches. Most importantly is the lack of interface standardization (most approaches, like LMS, use interface bridging). A standard approach is preferable to a customized approach. The non-FEM approaches provide some form of standardization but some of them are still not mature enough. The remaining choices include DCOM and CORBA. CORBA includes overhead such as complexity, expense, performance issues, and sheer size that may detract from its appeal. DCOM on the other hand is an integral part of the Windows operating system and is relatively easy to use for system integration. These and other reasons mentioned earlier indicate that DCOM is the most appealing architecture (of the approaches evaluated) for FEM-DSS interoperability.

An architectural design for FEM-DSS interoperability should satisfy several key requirements. The design should be a framework that is language-independent, so that applications can be implemented in different programming languages and used by clients that are written using different programming languages. The design should have extensibility, which allows application developers to customize a system to satisfy new needs. With extensibility, new components can be plugged into the current system, existing components can be replaced with new ones, or new components can be derived from existing ones. In this way, the current system grows and evolves over time, and components can be re-used by others.

Extensibility in turn requires that the design provide a generic set of interfaces; these interfaces enable new components to participate by extending them with minimal impact on current components. Common interfaces also form the basis for communications and interactions between components. The proposed architecture should provide a mechanism that enables communications and interactions. Furthermore, the design should be able to handle the peculiarities of legacy systems. For example, many legacy applications are not object-oriented, and data and functionality of one application may not be readily available to other applications, even if the applications are implemented using the same programming language and run on the same machine. There ought to be a standard facility that can be used by legacy systems to abstract and expose their services, so that these services can be understood and utilized by the rest of the framework. This often involves creating a so-called object wrapper that mediates between

a legacy system and other components. The easiest case is that only method interfaces need to be changed. However, the task will be much more difficult if the user interface of the legacy system needs to be modified and there is no documentation of its source code.

Finally, the design should be realistically applicable when implemented. The mission of the USDA Forest Service is to serve the general public and help manage national forests and associated ecosystems of the United States. Its public, non-profit nature implies that forest decision support systems developed by the agency are often distributed to the public freely or at nominal cost. As a result, any integration that presumably relies on costly commercial products will severely limit its use by the public.

6. A DCOM-Based Framework

Based on our evaluation, we propose a DCOM-based framework for the integration of forest decision support applications. This framework includes a typical programming language and development environment such as Microsoft Visual C++ or Visual Basic. Using NED-1 and FVS as example applications, our prototype demonstrates the effectiveness and appropriateness of integrating legacy and newly developed applications using DCOM. The implementation also indicates that, based on our previous experience with CORBA (Maheshwari 1997), DCOM programming is easier and more productive. This is because we only focus on the application-specific implementation while the framework does many routine tasks, for example, generating the templates necessary for creating DCOM objects and registering the applications. An additional advantage with DCOM is that we can develop distributed system interfaces using Microsoft resources. A slight difficulty with DCOM is that it is, after all, a technical specification that takes some time to master.

Figure 1 illustrates the general structure of our proposed DCOM-based architecture for integration. Conceptually, it has three major components: the caller or client requesting some service, the controller that has DCOM as its middleware, and the applications that provide the requested service.

The caller, for example NED-1, is an entity that issues a request to an application via the controller, and usually acts as an interface between the entire integrated system and the user. This interface is visual and may function differently than those of application objects in the system. It

may be the case that the caller is simply an initiator for the available applications and not have a corresponding application object. The caller can interact with one or more applications, for example FVS, to accomplish its work.

The controller is responsible for locating and activating applications. More importantly, it controls interactions between the caller and an application, and between applications. The controller uses DCOM as its backbone, because DCOM provides many system services that facilitate the registration of application components, finding the location of the application being requested, and the communications between the caller and the application. While running an application, the controller coordinates the callers' dialog with the user. An additional responsibility of the controller is to handle errors that may occur during an application's execution. If properly implemented, the entire process executes in a seamless way. Adding a new application to or replacing an existing one in the integrated system has minimal effect on how the framework looks to the user. Depending on the nature of the distributed system, the controller may also contain processing rules that help interpret the requests and instructions supplied by the user (via the caller). Therefore, using the controller, the applications participate in a coordinated fashion within the integrated information system.

An application is a component that provides services to the integrated system. For example, many of the forest decision support applications focus on simulation, display, input/output, or analysis tasks. Each application is encapsulated within an interface that follows a standard format (for example, the Microsoft Interface Definition Language, MIDL). This approach makes it possible for the application to communicate with the rest of the framework, such that other applications can use the interface to access the services this application provides. Interfacing also provides an effective way to deal with legacy applications. Many legacy applications were developed with a stand-alone purpose. Their data and functionality may not be readily available to other applications; certain interface modules of those legacy applications may be proprietary, limited, or even lacking. Newly constructed interfaces to the legacy applications act like adapters so that these legacy applications and the rest of the framework can work together, hence enabling re-use of existing applications.

The architectural design should be general purpose, meaning that the framework should have distributed processing capability and provide platform independence so that it is ready to

work in heterogeneous, cross-network environments if it is required to do so in the future. Overall, the proposed design is general and makes no assumptions about the software applications to be integrated. Its standardized interface scheme enables integration of a variety of applications. It is an open framework in the sense that application components can be added and/or removed without drastically affecting the functionality of the whole system. The adoption of DCOM as the middleware supports this design.

7. The NED-FVS Prototype

NED-1 is a full service, goal-driven ecosystem management decision support system (Twery et al. 1997, Rauscher et al. 1997). NED-1 is a multi-language system consisting of components written in both C++ and PROLOG. FVS is a widely used program for predicting vegetation dynamics over time (Teck et al. 1996). FVS is written in FORTRAN-77 as a DOS application. SUPPOSE is a Windows-based, user-friendly front-end to FVS written in C++ (Teck et al. 1997). NED-1 uses the functionality provided by SUPPOSE-FVS in order to forecast the consequences of implementing alternative management scenarios on a subject forest landscape. A description of the complete decision-analysis process supported by the NED-FVS prototype can be found in Rauscher et al. (in press). For the purposes of the present discussion, we can regard NED-1 as the Caller or Client program and SUPPOSE-FVS as a legacy application (see Figure 1). In this section we present a brief description of how we implemented the DCOM-based interoperable framework discussed previously.

The communication between the NED-1 client and any application module is through the controller (see Figure 2). In our prototype, we think of the NED-1 client and the controller as residing on the same computer system. Because of our use of the DCOM communication backbone, the design makes no assumptions about the physical location of any of the applications. The applications may reside on the local machine or on a remote machine without affecting the functionality of the system. Normally, the controller contains processing rules that help interpret the user's needs. These processing rules determine the kind of information that must be collected from the user. There is typically continuous interaction between the caller and the controller (Figure 2). The processing rules also determine which application to activate. In our NED-FVS prototype, the controller can be quite simple because it needs to know only about the SUPPOSE-

FVS system. Therefore, NED-1 issues a request to the controller to execute the SUPPOSE-FVS wrapper and then awaits a response from the controller. The response contains a message that either the task was successfully completed or that some error occurred. If an error occurred, NED-1 reports to the user the type of error otherwise the user is informed that a successful forecast has been made.

The controller executes the SUPPOSE-FVS wrapper. The task of a wrapper is to interpret the request of a calling client and manage an application program to implement a solution process. For example, the SUPPOSE-FVS wrapper knows how to understand requests from the controller and manage the SUPPOSE and FVS executable programs. The wrapper expects information from the controller about which Microsoft Access database file contains the relevant NED-1 source data and which database file is supposed to receive the predicted output data. The wrapper software contains the knowledge of how to proceed with the task. In our NED-FVS prototype, the wrapper calls a translator program (MDB-FVS) that finds the necessary information in a database file and creates the correctly formatted output files that both SUPPOSE and FVS require (Figure 2). The SUPPOSE-FVS wrapper then calls the SUPPOSE-wrapper.

The SUPPOSE-wrapper (Figure 2) checks to make sure the necessary support files are available and then executes SUPPOSE. SUPPOSE is a Windows program that directly communicates with the user to set up an FVS run by creating a keyword instruction file. This direct communication between a legacy application and the user is a compromise on our part. Ideally, only the NED-1 program should directly communicate with the user. But SUPPOSE is a complicated legacy application that would be difficult and expensive to duplicate, a situation that is frequently encountered. This compromise, however, necessitates that the SUPPOSE-wrapper must be more intelligent. It is quite possible for the user to create a keyword instruction file that fails to command FVS to create the expected output files in the expected format. To guard against this possibility, the SUPPOSE-wrapper must read this keyword instruction file, understand it, and fix any potential problems it encounters. If the wrapper finds it cannot fix a problem, it reports an error, stops the process, and reports this error back to the SUPPOSE-FVS wrapper that would then pass it along to NED-1 and the user. If the keyword instruction file passes inspection, then control passes back to the SUPPOSE-FVS wrapper.

Now, the SUPPOSE-FVS wrapper can execute the FVS wrapper along with the information about which FVS variant to use. The choices are: SOUTHEAST, NORTHEAST, SOUTHERN, or SOUTHERN APPALACHIAN. The wrapper executes the proper variant and checks that the expected output file has been created. Again, an error is reported if the expected output file cannot be found and control is passed back to the SUPPOSE-FVS wrapper.

The SUPPOSE-FVS wrapper then executes the FVS-MDB module that takes the FVS output file, finds the data of interest, and builds a new Microsoft Access database file containing the predicted growth and mortality information. Once the new database file exists, the SUPPOSE-FVS wrapper reports a successfully completed job to the controller and the controller reports the same to NED-1 and thru NED-1 to the user.

This description of the NED-FVS prototype highlights some important lessons of more general interest.

1. Wrappers are generally needed to manage the peculiarities associated with most legacy software.

2. Error checking and reporting is a necessary and important part of the process. Errors should be reported in the greatest detail possible in order to give the user and the other system components the best chance to correct them and obtain a successful run.

3. The organizational framework consisting of a controller-DCOM-wrapper-application sequence can be replicated into as many nested layers as needed in order to isolate software peculiarities and create a successful problem management system.

4. The generic nature of the communication framework need not be compromised by any peculiarities of the particular legacy software encountered.

5. The generic architecture facilitates the addition of other legacy and new applications since the infrastructure is already in place.

8. Discussion

To achieve more effective overall decision support for forest ecosystem management, it is necessary to integrate various existing decision support system modules. An interoperable

approach offers an interface standard that promotes communication between component modules and makes it possible to integrate newly developed modules and refine current modules over time, if necessary. More specifically, the approach should provide language interoperability, platform independence, inter-module communication mechanisms, object reusability, interface standards for dealing with legacy systems, and distributed processing power. It should not be restricted by any specific domain characteristics. It should serve only as a middleware framework. Using these criteria, we reviewed FEM-DSSs that try to solve the interoperability issue. Among them are NED-1, LOKI, FVS, LMS and INFORMS. Unfortunately, most of them adopt *ad hoc* approaches that are difficult to extend into generic interoperable architectures. LOKI appears to be a promising middleware framework, but it fails to pass all of the criteria. However, a new version of LOKI is under development.

In the meantime, we also investigated new technologies recently emerging (primarily during the mid-1990's) that deal with interoperability. They include the intelligent agent approach, DIAS/DEEM, CORBA, and DCOM. The agent approach is not concerned with low-level details about interoperability; using CORBA or DCOM as their respective facilitating middleware, most associated studies have focused on high-level interoperability such as semantics and complex coordination, with the majority of available agent environments being developed on UNIX platforms. The implementation of the approach tends to be expensive due to its technical complexity (Finin et al. 1994, Genesereth and Ketchpel 1994) and it has rarely been used in integrating real-world applications. Likewise, DIAS/DEEM is also a high-level framework. Designed to work in the UNIX environment and use CORBA as its communication middleware, it specifically deals with integration of environmental effects simulation models and therefore is not a general-purpose architecture. CORBA and DCOM are the two architectures that satisfy all of the criteria. They have mature specifications, are comparable in functionality, and both are being used widely in industrial, governmental and organizational projects. Because DCOM is embedded in Windows, it is considered a better choice for Windows-based applications (Grimes 1997).

After selecting DCOM, we set about developing a working prototype to integrate NED-1 and FVS. We developed a primitive controller to handle the interactions and a basic wrapper for the DOS version of FVS. We ran the prototype on a Windows95 machine; this meant that we had to install and set up DCOM. However, when this was done, the prototype worked very

smoothly. DCOM was totally transparent to the operation of the prototype. Although no performance tests were carried out, there appeared to be absolutely no performance degradation as a result of using DCOM. We are currently enhancing the prototype to integrate two other applications as well as have an intelligent query driven controller.

The controller is quite straightforward in our current implementation because of the simplicity of the coordinating activities it involves. However, if it needs to coordinate more complex communications between components for more applications, it may be a good idea to introduce a rule-based system into it (the next phase of the project addresses this issue). The future controller may also need to deal with database transaction processing for some applications. Our implementation marks the first phase of a long-term project to develop an intelligent information system for knowledge, data, and model management (Potter et al. 1992, Potter et al. 1994). As the first phase, security issues were not addressed. These issues will have to be addressed in the future in order to support our global distributed perspective. In addition, dealing with user interfaces of legacy applications that are to be DCOM servers deserves further investigation, especially if the servers are on remote machines. In the current implementation, the FVS server avoids the user interface of its legacy application by running a stand simulation in batch mode that requires little interaction with the user. However, in future phases of this project we will address the issue of legacy applications and how to transform then in an effective way so that they can be integrated within our architecture.

9. On The Horizon

New technologies are constantly being developed to improve existing computing approaches. The area of distributed computing is no exception. For example, during our investigations reported here, Microsoft, UserLand Software, and DevelopMentor collaborated on the development of a new interoperability approach called XML-RPC (Walsh 1998, Winer 1998, Udell 1999). Still in its infancy, the idea of using remote procedure calling over the Internet is gaining favor among "techies". XML-RPC may become the new standard for interoperability championed by Microsoft but it probably will take several years. For mainstream users that need a distributed interoperability solution today, DCOM provides the best approach. Of course, it

may not be the best approach in a few years but that's the way it goes in a high technology arena; change comes quickly.

Acknowledgement

References

Argonne National Laboratory. (1995a). The dynamic information architecture system: A high level architecture for modeling and simulation. Argonne National Laboratory, University of Chicago. *http://www.dis.anl.gov:80/DEEM/DIAS/diaswp.html*

Argonne National Laboratory. (1995b). DEEM high level architecture. Argonne National Laboratory, University of Chicago. *http://www.dis.anl.gov:80/DEEM/hla.html*

Bevins, C. D. and P. L. Andrews. (1993). The LOKI software architecture for fire & ecosystem modeling: A tinker toy approach. *Proceedings of the 12th Conference on Fire and Forest Meteorology*. pp. 252 259.

Bevins, C. D., P. L. Andrews, and R. E. Keane. (1995). Forest succession modeling using the LOKI software architecture. *Lesnictvi-Forestry* 41(4): 158-162.

Choo, Y. K. and C. C. Lee. (1997). Integrated distributed geographical information systems (IDGIS). *http://starr-www.tamu.edu/choo/papers/esri97.html*

Finin, T., R. Fritzson, D. McKay, and R. McEntire. (1994). KQML as an agent communication language. *Proceedings of the Third International Conference on Information and Knowledge Management*. ACM Press. pp. 1-8.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley. 395 p.

Genesereth, M. R. and S. P. Ketchpel. (1994). Software agents. *Communications of the ACM* 37(7): 48-53.

Grimes, R. (1997). *Professional DCOM Programming*. Birmingham, U.K: Wrox Press Ltd. 565 p.

Heiler, S. (1995). Semantic interoperability. *ACM Computing Surveys* 27(2): 271-273.

Jorgensen, S.E., B. Halling-Sorensen, and S.N. Nielsen. (1996). *Handbook of Environmental and Ecological Modeling*. Boca Raton, FL: Lewis Publishers.

Keane, R.E., D.G. Long, J.P. Menakis, W.J. Hann, and C. D. Bevins. (1996). Simulating coarse-scale vegetation dynamics using the Columbia River Basin succession model – CRBSUM. *General Technical Report INT-GTR-340*. Ogden, UT: USDA Forest Service, Intermountain Research Station.

Kramer, R., R. Nikolai, A. Koschel, C. Rolker, and P. Lockemann. (1997). WWW-UDK: A Web-based environmental meta-information system. *SIGMOD Record* 26(1): 16-20.

Leppinen, M., P. Pulkkinen, and A. Rautiainen. (1997). Java- and CORBA-based network management. *Computer* 30(6): 83-87.

Liu, S. (1998). *Integration of Forest Decision Support Systems: A Search for Interoperability*. Master's Thesis. Athens, GA: The University of Georgia. 122 p.

Lockemann, P.C., U. Kolsch, A. Koschel, R. Kramer, R. Nikolai, M. Wallrath, and H. Walter. (1997). The network as a global database: Challenges of interoperability, proactivity, interactiveness, legacy. *Proceedings of the 23rd VLDB Conference*. pp. 567-574.

Maheshwari, S. S. (1997). *A CORBA And Java Based Object Framework for Integration of Heterogeneous Systems*. Master's Thesis. Athens, GA: The University of Georgia. 92 p.

Manola, F. (1995). Interoperability issues in large-scale distributed object systems. *ACM Computing Surveys* 27(2): 268-270.

Mayfield, J., Y. Labrou, and T. Finin. (1996). Evaluation of KQML as an agent communication language. *Intelligent Agents Volume II - Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages*. M. Wooldridge, J.P. Muller and M. Tambe (eds). Lecture Notes in Artificial Intelligence, Springer-Verlag, 1996. 15 p.

McCarter, J. B., J. S. Wilson, P. J. Baker, J. L. Moffett, and C. D. Oliver. (1998). Landscape Management Through Integration of Existing Tools and Emerging Technologies. *Journal of Forestry*. June. pp. 17-23.

McCarter, J. B., J. S. Wilson, P. J. Baker, and C. E. Nelson. (1998). Landscape Management System User's Manual Version 1.6. LMS Project. College of Forest Resources, University of Washington. 97 p. *http://silvae.cfr.washington.edu/lms/lmsdown.htm*

Meek, B.L. (1994). Programming languages: Towards greater commonality. *ACM Sigplan Notices* 29(4): 49-57.

Microsoft. (1995). The Component Object Model Specification. Draft Version 0.9.
    http://premium.microsoft.com/msdn/library/specs/tech1/d1/s1d139.htm

Microsoft. (1996). The Component Object Model: Technical overview.
    http://www.microsoft.com/oledev/olecom/

Microsoft. (1997). Distributed Component Object Model Protocol - DCOM 1.0.
    http://premium.microsoft.com/msdn/library/techart/msdn_dcomprot.htm

Microsoft. (1998). DCOM: A business overview. *http://www.microsoft.com/oledev/olecom/*

Mowrer, H. T., K. Barber, J. Campbell, N. Crookston, C. Dahms, J. Day, J. Laacke, J.
    Merzenich, S. Mighton, M. Rauscher, K. Reynolds, J. Thompson, P. Trenchi, and M.
    Twery. (1997). Decision Support Systems for Ecosystem Management: An Evaluation of
    Existing Systems. *General Technical Report RM-GTR-296*. Fort Collins, CO: USDA
    Forest Service, Rocky Mountain Forest and Range Experiment Station. 154 p.

NC3A. (1997). NATO standards. *http://www1.nc3a.nato.int/NATOSTAN.HTM*

OMG. (1997). *The common object request broker: architecture and specification, Version 2.1*.
    OMG Document, Object Management Group.

Otte, R., P. Patrick, and M. Roy. (1996). *Understanding CORBA (Common Object Request
    Broker Architecture)*. Upper Saddle River, NJ: Prentice Hall.

Perisho, R. J., F. L. Oliveria, L. Forrest, and D. K. Loh. (1995). *INFORMS-R8 - A Tool for
    Ecosystem Analysis*. STARR Lab, Texas A&M University. 8 p.

Potter, W. D., T. A. Byrd, J. A. Miller, and K. J. Kochut. (1992). Extending decision support
    systems: The integration of data, knowledge, and model management. *Annals of
    Operations Research* 38: 501-527.

Potter, W. D., J.A. Miller, and K.J. Kochut. (1994). A Hyper-Semantic Approach to Intelligent
    Information Systems. *Integrated Computer-Aided Engineering*. 1(4): 341-357.

Rauscher, H. M. (1995). Natural resource decision support: Theory and practice. *AI
    Applications* 9(3): 1-1.

Rauscher, H. M. (1999). Ecosystem management decision support for public forests: A review.
    Forest Ecology and Management. 114: 173-197.

Rauscher, H. M., R. P. Kollasch, S. A. Thomasma, D. E. Nute, N. Chen, M. J. Twery, D. J.
    Bennett, and H. Cleveland. (1997). NED-1: A goal-driven ecosystem management
    decision support system: Technical description. *Proc. of GIS World '97*. pp. 324-332.

Rauscher, H. M., F. T. Lloyd, D. L. Loftis, and M. J. Twery. (in press). A practical decision-analysis process for conducting ecosystem management at the project-level on National Forests of the United States. *Computers and Electronics in Agriculture*.

Reynolds, K., Saunders, M., Miller, B., Murray, S. and Slade, J., (1997). An application framework for decision support in environmental assessment. In: *Proceedings of GIS World '97, Integrating spatial information technologies for tomorrow*, 17-20 February 1997, in Vancouver, BC, pp. 333-337.

Sheth, A. (1998). Changing Focus on Interoperability in Information Systems: From System, Syntax, Structure to Semantics. *Interoperating Geographic Information Systems*. M.F. Goodchild, M.J. Egenhofer, R. Fegeas, and C.A. Kottman (eds). Kluwer Pub. Co.

STARR Lab. (1993). INFORMS-TX Users Guide. *INFORMS-TX Version 1.0*. STARR Lab, Texas A&M University.

Teck, R., M. Moeur, and B. Eav. (1996). Forecasting ecosystems with the forest vegetation simulator. *Journal of Forestry* 94(12): 7-10.

Teck, R., M. Moeur, and B. Eav. (1997). The forest vegetation simulator: A decision-support tool for integrating resources science. *http://www.fs.fed.us/ftproot/pub/fmsc/fvsdesc.htm*

Twery, M.J., D.J. Bennett, R.P. Kollasch, S.A. Thomasma, S.L. Stout, J.F., Palmer, R.A. Hoffman, D.S. DeCalesta, J. Hornbeck, H.M. Rauscher, J. Steinman, E. Gustafson, G. Miller, H. Cleveland, M. Grove, B. McGuinness, N. Chen, and D. E. Nute. (1997). NED-1: An integrated decision support system for ecosystem management. *Proceedings of the Resource Technology '97 Meeting*. pp. 331-343.

Udell, J. (1999). Exploring XML-RPC: Web-based Distributed Computing Gets Even Easier. *BYTE.com*. *http://byte.com/features/1999/06/0607XML_RPC.html*.

von Bultzingsloewen, G., A. Koschel, and R. Kramer. (1996). Active information delivery in a CORBA-based distributed information system. *Proceedings of First IFCIS International Conference on Cooperative Information Systems*. pp. 218-227

Walsh, J. (1998). Microsoft Spearheads Protocol Push. *InfoWorld Electronic*. *http://www.infoworld.com/cgi-bin/displayStory.pl?980710.whsoap.htm*

Wegner, P. (1996). Interoperability. *ACM Computing Surveys* 28(1): 285-287.

Winer, D. (1998). XML-RPC For Newbies. *http://davenet.userland.com/1998/07/14/xmlRpcForNewbies*
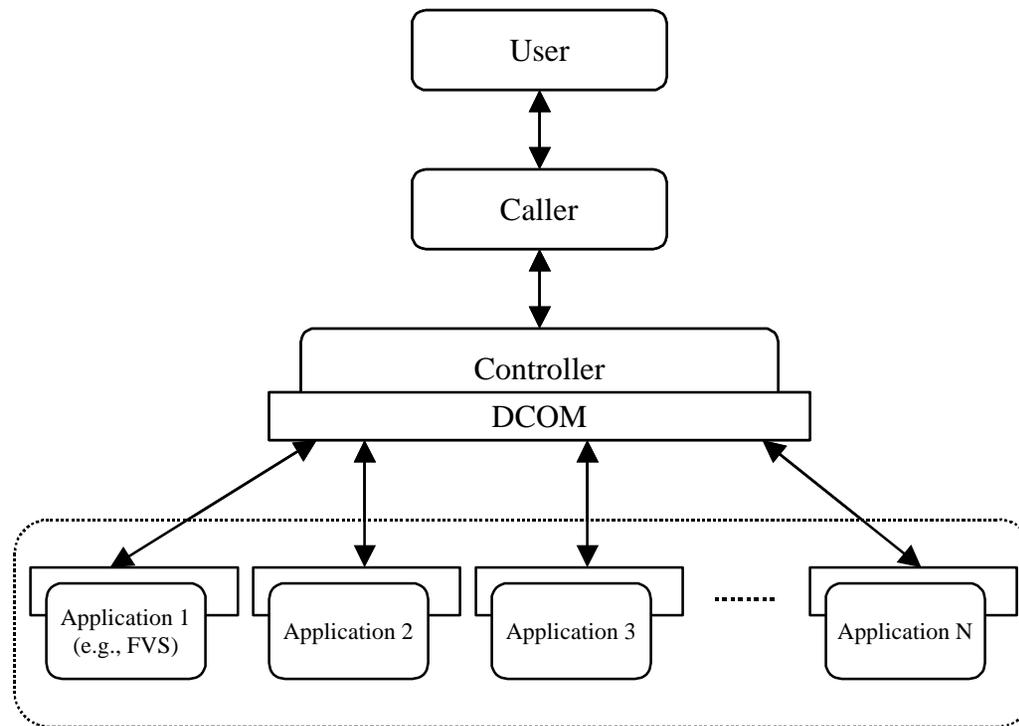
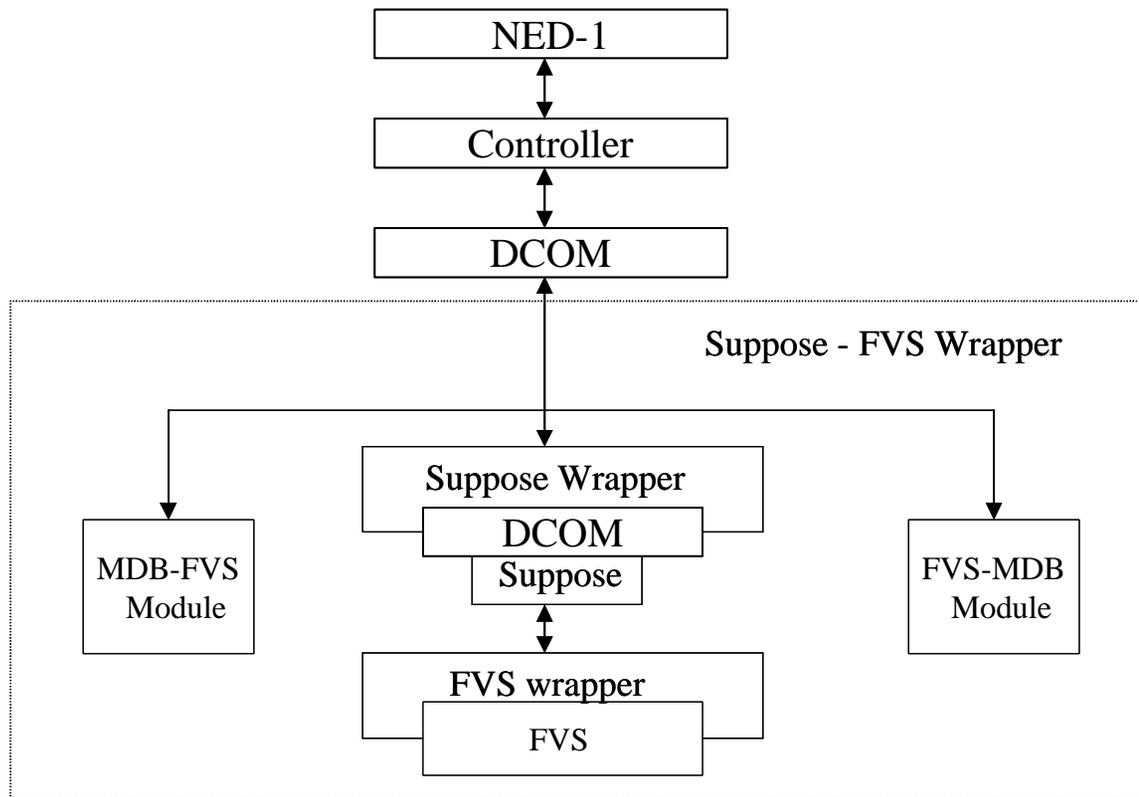Figure 1. The DCOM-based interoperable architecture.

Figure 2 Flow Chart of NED - FVS Application

Table 1.   Comparison of NED-1, LOKI, FVS, INFORMS, and LMS

CRITERIA \ APPROACH	NED-1	LOKI	FVS	INFORMS	LMS
Purpose	Goal-driven, Full Service DSS	Middleware (claimed)	Vegetation Dynamics	Data-driven, Full Service DSS	Vegetation Simulation Analysis and Visualization
Level	High	Medium	High	High	High
Domain Limited	YES – Forest Management	Unknown	YES – Vegetation Dynamics Simulations	YES – Forest Management	YES – Forest Management
Language Interoperability	NO	YES (claimed)	NO	NO	NO
Platform:					
Workstation/UNIX	NO	YES	YES	YES	NO
Personal Computer	YES	NO	YES (MS-DOS Mode)	NO	YES
Independence	NO	NO	NO	NO	NO
Module Coordination	Ad hoc – Point to point	Lack of Detailed Info	Ad hoc – Point to point	Ad hoc – Point to point	Ad hoc – Point to point
Distributed Processing	NO	YES (claimed)	NO	NO	NO
Object Reuse	Source Code	Unknown – Relevant Information Unavailable	Not Object-based	Not Object-based	Not Object-based
Dealing with Legacy Systems	NO	Presumably NO	NO	NO	YES, But No Standard Mechanism
Internet Supported	NO	NO	NO	NO	NO
Status	Operational – Continues To Add New Functionality	Prototype – In Re-write, Not Available	Operational for Many Regions	Operational	Operational
Cost	NONE	Wait to See	NONE	NONE	NONE
Interoperability Documentation	Some	Very little	Some	Some	Fair
Special Remark	OO Paradigm	In-house research project under development	Powerful post-processing capabilities	Uses FVS for vegetation simulations	Several linked vegetation simulations

Table 2.   Comparison of CORBA, DCOM, Intelligent Agent-based Approach, and DIAS/DEEM

APPROACH / CRITERIA	CORBA	DCOM	Agent Approach	DIAS/DEEM
Purpose	Middleware	Middleware	Distributed Computing, Semantic Interoperability Autonomy	Model Integration
Level	Low	Low	High	High
Domain Limited	NO	NO	NO	Environmental Applications
Language Independence	YES – Has Well-defined Mapping Mechanism (Interfacing using OMG IDL)	YES – Has Well-defined Mapping Mechanism (Interfacing using MIDL)	YES – Via ACL	YES, But Has No Well-defined Mechanism
Platform: Workstation/UNIX Personal Computer Independence	YES YES YES	YES YES YES	YES NO Varies	YES NO NO
Module Coordination	General	General	Varies	Ad hoc – Point to Point
Distributed Processing	YES – Has Well-defined Transport Mechanism	YES – Has Well-defined Transport Mechanism	YES – Uses CORBA, DCOM or Other Protocols	YES – May Use CORBA In the Future
Object Reuse	Source Code	Binary (Executable Code)	Varies	Source Code
Dealing with Legacy	ORB Interfacing	DCOM Interfacing	Varies	YES – Not Well-defined
Internet Supported	YES – Through IIOP	YES – Through ActiveX	YES (for Most Systems)	NO
Status	Mature Standard – Implemented by Multiple Vendors	Mature Standard – Implemented by Microsoft	Conceptual/Prototype	Prototype
Cost	High - $1,000 or More	No Up-front Cost – Free Download or Shipped with New Windows OS	Varies – High for Many if Implemented	No Information
Technical Documentation	Well-documented – On-line, Books, and Journal Articles	Well-documented – On-line, Books, and Journal Articles	Many lack detailed Documentation	Not Well-documented
Special Remark	Dominates UNIX	Wedding to Windows OS	Promising in Theory	Nice ES-based Context Manager