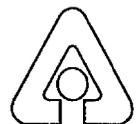

OpenLink: A Flexible Integration System for Environmental Risk Analysis and Management

**Environmental Assessment Division
Argonne National Laboratory**

Operated by The University of Chicago,
under Contract W-31-109-Eng-38, for the

United States Department of Energy



Argonne National Laboratory

Argonne National Laboratory, with facilities in the states of Illinois and Idaho, is owned by the United States Government and operated by The University of Chicago under the provisions of a contract with the Department of Energy.

This technical memorandum is a product of Argonne's Environmental Assessment Division (EAD). For information on the division's scientific and engineering activities, contact:

Director, Environmental Assessment Division
Argonne National Laboratory
Argonne, Illinois 60439
Telephone (630) 252-3107

Presented in this technical memorandum are preliminary results of ongoing work or work that is more limited in scope and depth than that described in formal reports issued by the EAD.

Publishing support services were provided by Argonne's Information and Publishing Division (for more information, see IPD's home page: <http://www.ipd.anl.gov/>).

Disclaimer

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor The University of Chicago, nor any of their employees or officers, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of document authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof, Argonne National Laboratory, or The University of Chicago.

Available electronically at <http://www.doe.gov/bridge>

Available for a processing fee to U.S. Department of Energy and its contractors, in paper, from:

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
phone: (865) 576-8401
fax: (865) 576-5728
email: reports@adonis.osti.gov

ANL/EAD/TM-114

OpenLink: A Flexible Integration System for Environmental Risk Analysis and Management

by D.J. LePoire, J. Arnish, E. Gnanapragasam, T. Klett,
R. Johnson, S.Y. Chen, B. Biwer, and C. Yu

Environmental Assessment Division
Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439

October 2001

Work sponsored by U.S. Department of Energy



This report is printed on recycled paper.

CONTENTS

ABSTRACT	1
1 BACKGROUND AND INTRODUCTION.....	1
2 OPPORTUNITIES AND GOALS	2
3 TECHNOLOGY ASSESSMENT	5
3.1 Criteria and Issues	5
3.2 Options	6
3.3 Proposed Solution	9
3.3.1 Layers.....	9
3.3.2 Roles.....	11
3.3.3 New and Existing Software Development	12
4 TECHNOLOGY DEMONSTRATIONS.....	13
4.1 Components.....	13
4.1.1 Models.....	13
4.1.2 Presentation	15
4.1.3 Data	16
4.1.4 Network.....	16
4.1.5 Integration and Connections.....	17
4.2 Integration Demonstration Projects.....	17
4.2.1 Low-Level Landfill Analysis with DUST and RESRAD	18
4.2.2 MARSSIM Analysis with RESRAD	20
4.2.3 Nuclide Web Service.....	24
4.3 Future	26
5 CONCLUSIONS.....	28
5.1 Demonstration Assessment Versus Criteria.....	28
5.2 Technology Uncertainties and Risks.....	29
5.3 Organizational Uncertainties and Risks	30
6 RECOMMENDATIONS	32
7 REFERENCES.....	35

FIGURES

1	Goals for the Open Environmental Modeling System, OpenLink.....	10
2	Four of the Five Layers of the Modularization Approach to Model Development	10
3	Roles of Those Who Develop and Use System.....	12
4	Integrating Models, Data, and User Interfaces.....	14
5	Integrating RESRAD-OFFSITE and DUST	19
6	Constructing a Custom Interface.....	20
7	Integrating User Interface Forms into RESRAD-OFFSITE Interface	21
8	Wrapping RESRAD for MARSSIM Analysis.....	22
9	Integrating the RESRAD Component to an Interface Input Web Page	23
10	Connecting Graphics Packages to Results Database.....	24
11	Developing and Using a .NET Nuclide Web Service	26
12	Constructing a Simplified Object-Oriented Version of RESRAD.....	27
13	OpenLink Sharing of Models, Data, and Interface Components	33
14	OpenLink's Dynamic Feedback System.....	33

TABLE

1	Criteria for Three System Architectures	7
---	---	---

OPENLINK: A FLEXIBLE INTEGRATION SYSTEM FOR ENVIRONMENTAL RISK ANALYSIS AND MANAGEMENT

by

D.J. LePoire, J. Arnish, E. Gnanapragasam, T. Klett,
R. Johnson, S.Y. Chen, B. Biwer, and C.Yu

ABSTRACT

Most existing computer codes for modeling environmental pathways were developed to satisfy a specific objective (e.g., perform analyses to demonstrate regulatory compliance). Over time, the codes have been written in various computer languages and software environments that are often not compatible with each other. In recent years, largely driven by advances in industrial software, a new concept for software development based on modularization has emerged. This approach entails the development of common modules or components that can be shared by and used in different applications that have certain common needs. Although modularization promises advantages over the traditional approach, a number of issues must be fully addressed and resolved before the approach can be accepted as a new paradigm for environmental modeling. This report discusses these issues, provides demonstrations of open integration techniques, and provides recommendations and a course of action for future development.

1 BACKGROUND AND INTRODUCTION

Most existing computer codes for environmental pathway modeling were developed to satisfy a specific objective (e.g., perform analyses to demonstrate regulatory compliance). Over time, the codes have been written in various computer languages and software environments that are often not compatible with each other. In recent years, largely driven by advances in industrial software development, a new concept for software development based on modularization has emerged. This approach entails the development of common modules or components that can be shared by and used in different applications that have certain common needs. For instance, an air dispersion model could be written into a common component to be shared by several different applications, each with the need to model air dispersion of some release. When fully developed, the modeling application would become an exercise of selecting, integrating, and applying a consistent combination of appropriate modules for a specific problem. Although modularization promises advantages over the traditional approach, a number of issues do exist. These issues must be fully addressed and resolved before the approach can be accepted as a new paradigm for environmental modeling. This report discusses these issues and provides recommendations and a course of actions for future development.

Traditionally, model connections have been made by the end user, who aligned one model's output data with another model's input. Often the model assumptions and conceptualizations were stretched to accomplish the linkage, resulting in greater uncertainty in the results. Also, the connection usually required the user to invest effort in manipulating the data for proper communication (e.g., taking data from the first model's output and manually editing the input for the next level), resulting in inefficient use of resources and introducing another potential source of error. It is generally difficult to connect models because of their disparate assumptions about scale, conceptualization, aggregation, process, reality, and objectives. Systems in other disciplines have been developed by using function libraries and toolboxes to prepare and manipulate data.

2 OPPORTUNITIES AND GOALS

In the environmental field, modeling plays a critical role in connecting current data and knowledge with predictions of future events and environmental states. Environmental problems are quite challenging to solve because of the complex relationships among many contributing factors, both natural and man-made (Constanza et al. 1993). Moreover, these problems need to be addressed not only by environmental engineers and regulators but also by concerned members of the public and nongovernmental organizations. Their demands on environmental modeling often conflict because predictions need to be accurate yet easily understood, communicated, and explored. The increasing complexity of environmental codes also places a demand on the end user, who must translate the real environmental problem into the conceptualization allowed by the model and its options. Information on assumptions and options must be conveyed to the user to ensure that the model is applied and interpreted correctly. Open communications about the model, interface, and data components would enable software applications to be more easily developed.

The complex and conflicting goals placed on environmental modeling are illustrated in the U.S. Nuclear Regulatory Commission's (NRC's) recently stated goals of an environmental modeling system:

- Maintain safety and protection of the environment,
- Increase public confidence,
- Increase efficiency and effectiveness of decisions, and
- Reduce unnecessary burdens on stakeholders.

Many modeling approaches could be used to realize these goals. For some purposes, the most detailed models and data are appropriate to predict a situation. However, the results from these models might not facilitate a good understanding of the situation, or they might place an undue burden on the stakeholders as they try to learn how the models work and they try to gather sufficiently verified data. Moreover, the results might not be directly integrated into regulatory processes, so an interpretation of the results, although accurate, would be difficult to accomplish.

Another approach is to use simplified models with conservative values for data to facilitate an understanding of the results and to place bounds on them. This approach enables the end user or interpreter to focus on the important issues. Many regulatory processes include this type of analysis to help users explore the issues and decide whether a more detailed analysis is justified.

There is a wide gap between these two approaches and between their contributions to understanding and the decision-making process. Both can be enhanced with tools that allow sensitivity analysis, uncertainty analysis, and visualization and manipulation of the data.

When these considerations are taken into account, it might be reasonable to expect that a range of models and also a range of integration strategies are available. On the detailed side of modeling, the emphasis might be placed on manipulating large, standardized data sets covering spatial and temporal dimensions. The emphasis on the user interface and regulatory requirements could be less, since the end user would be assumed to be more cognizant of the assumptions and therefore more responsible for constructing valid model input. Other times it might be important for the user to just to “scope out” a situation and determine the major exposure and risk pathways. This situation might be enhanced by a very flexible system that would allow the end user to easily explore the simplified models. Another emphasis might be on the need to develop some components that could be used in detailed and mid-level modeling environments that require intermediate modeling sophistication and in guiding the user through the regulatory process. These different modeling needs might require some shared models but different integration environments. This requirement could be met by developing various user interface components and standards, process flow standards, data management and sharing tools, decision support tools, and model linkages so that these models could become more flexible, efficient, and effective.

Not only are the demands for environmental modeling changing, but also the means to accomplish this modeling are being developed and quickly changing. Many of these means to accomplish modularization and integration are based on the supply of new information technologies (ITs) originally developed for businesses. IT technologies, such as those for object-oriented code development, network-based distributed processes, database storage and manipulation, graphical user interfaces (GUIs), work-flow processing and communication, geographic information systems (GISs), and graphical visualization, have become available for integration into environmental modeling in recent years. These tools and techniques, which required large investments by commercial vendors, are available relatively inexpensively because of the demand from the business community. Software packages for environmental modeling should take advantage of these tools and techniques in responding to the field’s unique needs and demands. The emphasis should not be on redeveloping industry standards but on meeting environmental modeling’s unique needs by leveraging these tools and techniques.

3 TECHNOLOGY ASSESSMENT

3.1 CRITERIA AND ISSUES

If a next-generation risk-modeling environment is to be successful, it must address a range of needs and issues. Although a single integration platform (fixed-platform) approach offers a step forward in addressing modularization needs, it also has a number of shortcomings that must be addressed for it to be successful and gain widespread acceptance. Identified issues related to the fixed-platform system are discussed in the following list.

- *Flexibility and maintenance:* The rigidity of the framework allows little flexibility for incorporating “foreign” components not conforming to the framework specifications; that is, the fixed-platform has difficulties interacting with components developed outside platform specifications. The requirement to model a range of conceptualizations for a range of user needs leads to the demand for a more flexible system.
- *Software dissemination:* The fixed-platform system and its components are installed separately and cannot be distributed in a single package for application. Barriers to the dissemination and installation of the code package (data, software, and documentation) must be low to ensure a large base of users who can leverage their common understanding and share their experiences. Software dissemination should also be tied to Internet maintenance for a user community.
- *Quality assurance (QA):* The QA of individual components can be done by the developers, but the user has to bear the burden of ensuring that the QA of the integrated application is properly done. For legacy (existing) computer codes, significant effort has to be expended to convert these codes into the fixed-platform system in order for them to be functional. The fixed-platform system allows no flexibility in this regard. This particular drawback could potentially hinder interagency collaboration in terms of committing future resources for code conversion. This issue also relates to the system’s ability to be validated and verified.
- *Life-cycle development and maintenance costs:* Long-term cost savings might be anticipated from using this approach rather than the traditional development approach. However, the total life-cycle development and maintenance costs would depend on future needs and are yet to be evaluated and assessed. More effective and efficient approaches (discussed in the following section) might become feasible, thereby rendering the fixed-platform obsolete.
- *Platform reliance:* The rigidity of the platform structure would compel future code developers to rely totally on the system, thereby discouraging competing

existing software to be included. Additionally, since the system offers no “future-proof” assurance, its technology would be prone to change in the future. This would be a serious flaw for federal agencies interested in investing in a long-term, stable system.

- *Transparency:* This issue is tightly bound to the user interface and documentation and sometimes conflicts with the complexity of the site being modeled and realistic goals. To be transparent, both the model itself and its implicit and explicit assumptions must be understandable. Many times, detailed, complex, realistic models are difficult to understand. Sensitivity analysis helps but can also obscure combinations of parameters that are driving the results. The level of the model’s conservatism versus realism should also be clear. Such knowledge affects public acceptance and confidence if the model is to be publicly used or defended.
- *Ease of use:* There are different requirements for ease of use, depending on the task and frequency of use. Some applications might be infrequently used and need to implement a specific process for regulations. Other applications might be designed for a relatively inexperienced user to explore a problem. Still others might demand in-depth knowledge of the assumptions and might be applied quite frequently. After these factors are taken into account, there is a set of general heuristics to consider when developing and designing user interfaces.

3.2 OPTIONS

Various options are available for implementing a more flexible environmental modeling environment. These include (1) continuing with the current status quo approach, (2) adopting a single model-, data-, and user-interface-integrated framework, and (3) using separate tools to integrate models, data, and interfaces. On the basis of our experience and discussions about three types of software architecture, we created a table listing the criteria and attributes of each architecture (Table 1).

In the current approach, features, tests, and documentation are added in a piecemeal fashion. The interface, data, and model are integrated for a specified set of objectives. The end user’s experiences and contexts are considered when the model options, interface, and result visualizations are being designed. This process tends to encourage code that is unwieldy, as new features are added without the code being redeveloped and modularized. Software breakdown is not similar to mechanical breakdown (i.e., initial break-in period, period of stable performance, and then mechanical wear), because software does not wear down or change. However, by introducing changes to the software, undesired entropy can be easily introduced into the original design. Technologies in the software application environment can change, causing more effort to be expended just to maintain operation of the software. This situation does not necessarily occur in object-oriented software. Reliance on some commercially developed tools can lead to a need

TABLE 1 Criteria for Three System Architectures

Criteria	Current Approach	Open Architecture	Specified Visual Programming Framework
Description	Features, tests, and documentation are added in piecemeal fashion. Interface, data, and model are integrated for a specified set of objectives.	Development and testing are divided into three levels: modules, integration, and end use. This approach allows module reuse and swapping and provides the ability to develop flexible end-user interfaces and data management.	Development and testing are divided into two levels: (1) modules and (2) end-user integration and implementation through a single visual programming framework.
Maintainability	Since modular design was not the main focus, it is sometimes unwieldy to integrate and test new functions.	Modules are maintained by the developers. Standards are agreed to and followed in module and data specification. Integration can be done in a number of ways depending on the requirements.	The framework must have one specified standard. All codes must go through the standard to be incorporated.
Dissemination	At user's initiation to download and install	Modules can be distributed with the integrated application. Later modules can be maintained on distributed servers.	Framework and modules are installed separately.
Validation and verification (V&V)	Verification is done for a complete specific version and maintained incrementally. Validation can be done on limited aspects of the model.	Each module maintains its own V&V. Applications connecting the modules are done by integrators who ensure assumptions are appropriately compatible for the application V&V.	Modules can be V&V'd, but V&V of the integration process is up to the end user.
Flexibility	Adding new features and functionality is difficult, requiring changes throughout the code and careful consideration of many of the obscure details of the code.	Modules can be added, substituted, and modified with flexible connections to other modules. This practice allows for flexibility in both the module level and the integration level.	Modules can be added as long as they fit the framework's fixed structure. Modules cannot be flexibly integrated for other potential integrating frameworks.
Use of legacy software	It is very difficult to integrate two legacy codes and maintain the assumptions and user interface.	Legacy code can be "wrapped" for use with other codes. Some functions can be called separately. A modularized version of the model would be more flexible.	It is difficult to incorporate legacy code without a large effort to modularize it to conform to the framework's fixed structure.
Support for cooperation	It is very difficult to maintain one code that serves two agencies with different requirements.	Modules and data can be shared for different applications. Different applications can be constructed with the shared modules to accommodate the different requirements of the agencies.	All agencies can develop their own modules, but they must conform to the framework structure. It may be difficult to construct one structure to satisfy the needs of all agencies and organizations.
Development costs	Development is inefficient because new features must be highly customized for each application.	Modularization and structural flexibility lead to efficient reuse and development of modules while maintaining an efficient user interface.	Modularization leads to more efficiency, but effort can be expended on conforming the modules to a structure that is not efficient and effective.

TABLE 1 (Cont.)

Criteria	Current Approach	Open Architecture	Specified Visual Programming Framework
User interface support for regulatory processes	Support is good since the user interface can be tailored to control the input and reporting process.	The flexibility of the connections allows process and user interfaces to be well defined.	The end user can gain understanding through quick exploration of models; however, to implement regulatory processes, the user must invest effort to visually program the process.
Platform independence	Platform is dependent on the operating system (OS) and developer's tools. Development can be hindered by uncontrolled changes by the supplier of the OS and integrated development environment (IDE).	The modules can be platform independent. The connections between them are flexible and can be implemented with many technologies.	Platform is very dependent on the structure of the framework. Development can be hindered by incompatibilities of the model or process to be used and the framework.

to constantly upgrade even well-written software, just to maintain its operation in a changing technological environment.

In a single integration framework, development and testing are divided into two levels: (1) development of modules and (2) end-user integration and implementation through a single visual programming framework. This framework works as long as it is flexible enough to meet various needs. However, it is very difficult to leverage new technology within the framework, since the user-interface, data manipulation, and modeling connections are already specified and implemented. This system can facilitate the exploration of a specific environmental problem by a single end user but can cause difficulties for a user community whose members are trying to follow a regulatory process. Also, the burdens of model integration and application are on the end user. (Frames 1.1 and GoldSim [Whelan et al. 1997; Whelan and Nicholson 2001] are examples of this type of system.)

Quite a large set of tools is being developed to further separate the roles of modelers and integrators and the four components (data, models, interface, and connection). Some model integration tools include the Argonne National Laboratory (ANL) DIAS system (Sydelko et al. 1999) and the U.S. Environmental Protection Agency (EPA) MIMS system. These tools offer a system of utilities for model integration and data communication. The DIAS tool is based on the concept of using models to provide methods for a higher-level conceptualization of an object. This allows both new development and the wrapping of existing models. However, there are many other ways to accomplish this wrapping and object integration with commercial tools (J2EE, ColdFusion [Forta 1998], Microsoft [MS] .NET [Hollis and Lhotka 2001]) that might not supply the same utility support but allow a flexible integration with commercial components.

Besides these model integration tools, there are also tools that allow user interface and data management module reuse and swapping. The user interface and visualization aspects are crucial and difficult to construct with automated tools. The interface usually requires a great deal

of customization to ensure that users understand the data, options, work flow throughout the process, and model assumptions.

One commercial system that seems to have a good approach is the Environmental Systems Research Institute ArcIMS system (ESRI 2001), an Internet-based system for supplying GIS maps and data. The main map-rendering application is deployed on a server. The developer works with this service and is supplied with a default set of tools to develop a user interface for the manipulation and display of the maps. The interface components are object-oriented but written in a client scripting language (JavaScript). This allows the component provider (ESRI) to provide a flexible template to the integrator to customize the user interface for the end user.

3.3 PROPOSED SOLUTION

The proposed solution includes developing strategies and guidelines for separating the software package components into a set of layers and identifying roles for model development and use. The strategies can apply to both the modification of existing codes and practices and the development of new models and components.

3.3.1 Layers

Like the traditional approach toward code development, the modularization approach for developing a complete modeling package usually consists of five generally distinct elements (Figures 1 and 2):

1. *Application layer:* Integrators form specific combinations of components that apply to a range of end-user purposes yet still facilitate understanding or demonstrate compliance. The components are selected and connected from modeling, data, and interface support.
2. *Presentation layer:* This user-friendly interface facilitates understanding through guided input and presentation of results.
3. *Model layer:* Models are encoded by developers to describe physical phenomena. Other models might be used to control the functionality of the calculations (e.g., uncertainty analysis).
4. *Data layer:* Data are collected to support parameters and analyses that are used in models. Issues to be addressed include how to store and transfer the data that the models need.
5. *Network layer:* This layer connects distributed data and components and supplies the communication tools to integrate them.

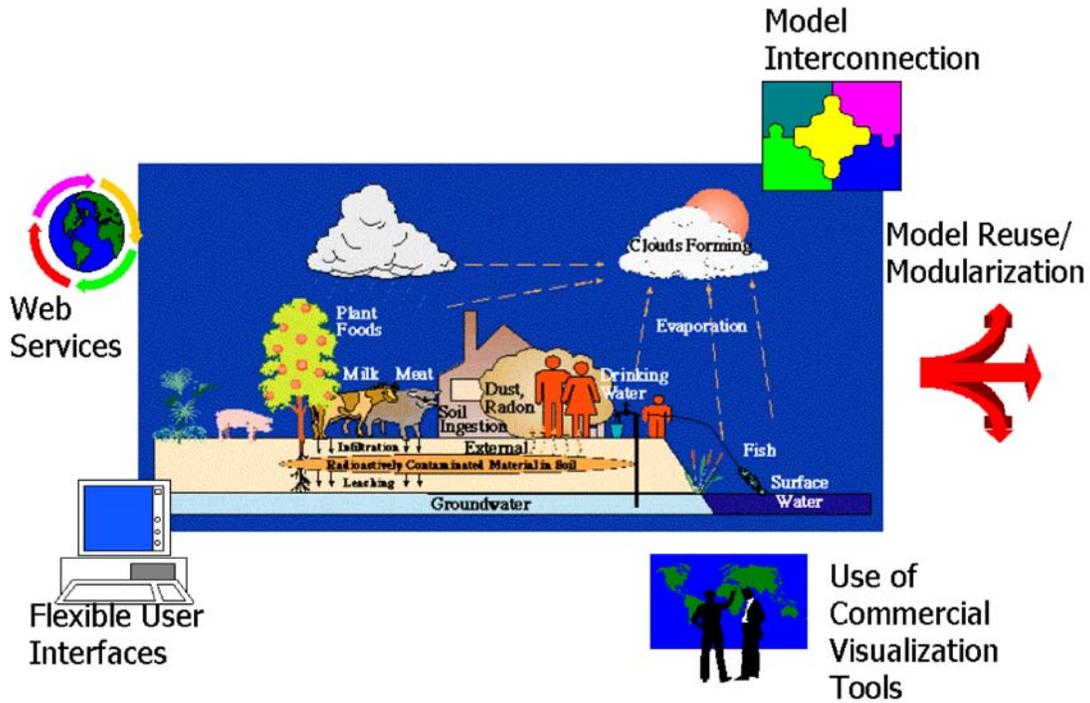


FIGURE 1 Goals for the Open Environmental Modeling System, OpenLink

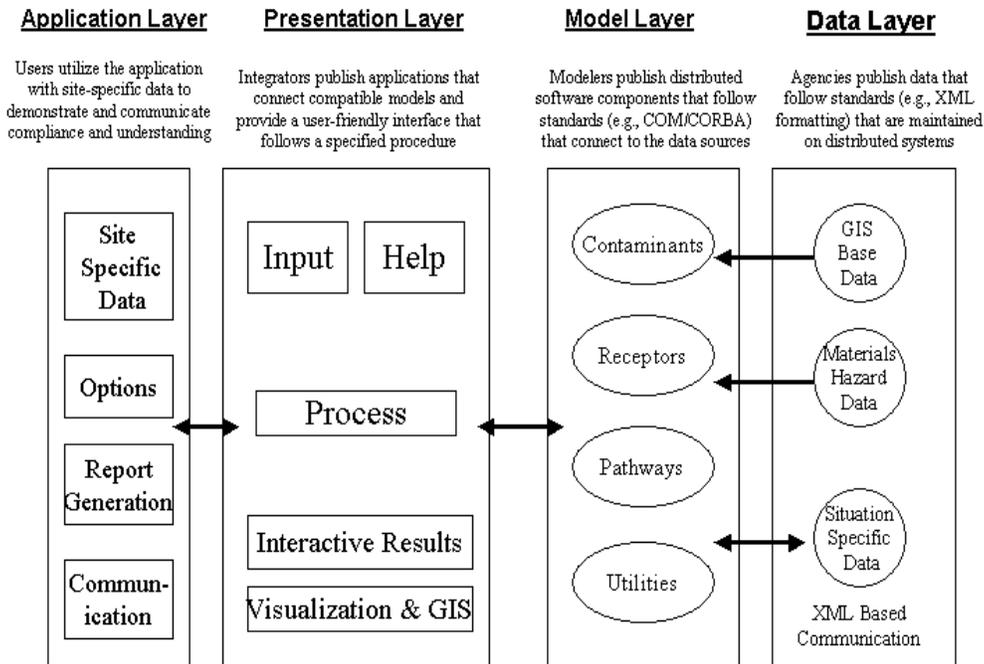


FIGURE 2 Four of the Five Layers of the Modularization Approach to Model Development (Components can be developed in the presentation, model, and data layers. They are integrated in the application layer by a variety of tools and techniques.)

While the traditional method integrates these elements into a single code, the modularization approach instead aims at building a code system consisting of components that can be used and reused for various purposes.

Over the last six years, a group of agencies (U.S. Department of Energy [DOE], NRC, EPA, U.S. Department of Defense [DOD], and others) has been informally discussing the problems in linking models for complex simulations of the environment. This group met in March 2000 for the Environmental Software Systems Compatibility and Linkage Workshop (Whelan and Nicholson 2001). The group has continued its work, and members attended a meeting in June 2001 to discuss understandings, accomplishments, and future paths. One objective was to establish categories of attributes for software systems. These included a set somewhat similar to the standard layers described above:

- Model connectivity (model layer),
- Information architecture (data layer),
- Framework connectivity (presentation layer),
- Web-based access (network layer), and
- System functionality (application layer).

3.3.2 Roles

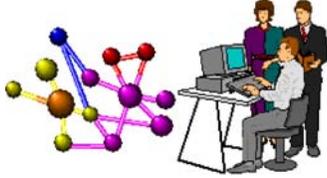
Three sets of roles are proposed for developing and using the system (Figure 3). First, modelers should develop domain-specific models and document their assumptions. Second, integrators should create an application from the available models and data. The integration environment would be up to the integrator (i.e., there would be no single integration framework, so the system could be done in a web environment [e.g., Active Server Pages or ColdFusion], as a window standalone, or as a hybrid using web services). Third, end users should then specify the data and options through the integrated user interface and communicate the results to the regulators and public.

The models should be available for all to use with technology like an application programming interface (API) for a modular class library or like a set of services (data and model) from a server. Use of generic commercial technology would allow components developed in different laboratories with different software languages to be integrated. This coordination would require the laboratories to work cooperatively in defining appropriate scales, aggregations, and assumptions. The effort would include the development of new models and data and the opening of existing codes.

To maintain the most flexibility and cost-effectiveness, the integrators should leverage commercial technologies. There are quite a few commercial tools to support these tasks, and the technology is rapidly changing, being mostly driven by business software. These tools can be



Modelers construct models and publish assumptions, input required, and available functions. Source code is not necessarily published.



Integrators construct applications from available models and data to implement regulatory processes with an easy to use interface.



End Users for specific sites. Can also involve managers, decision makers, regulators, and the public.

FIGURE 3 Roles of Those Who Develop and Use System (Components can be developed by modelers who specify the assumptions and QA the models. The components are integrated for a specific application by integrators who QA the integrity of the model assumptions. End users select model options and conceptualize their cases and sites.)

utilized in these systems so that the focus can be on the models and data and on facilitating user understanding for effective decision-making.

3.3.3 New and Existing Software Development

In this paradigm, an interagency committee specifies the standards for communicating between codes and establishes the minimum validation and verification (V&V) process required for individual components. Code developers at specific agencies or institutions maintain control over their own sources but use the specified standards for “exposing” their codes for inclusion in other applications. While this approach would work best with new code development, methods for “wrapping” legacy code that allow reuse of existing code are available. This structure would make it possible to embed distributed components within everything from simple spreadsheet applications, to commercial GIS packages such as ArcView™, to a visual programming environment, if that was desired.

Taken one step further, components for modeling, presentation, and data access could be maintained at remote locations through the use of MS COM objects or Java servlets. Their integration could be accomplished with software tools such as CORBA, DCOM, and RMI. Communication between components could be accomplished with APIs using evolving standards such as Interface Definition Language (IDL) and eXtensible Markup Language (XML). The use of distributed objects would even make it possible for object components and data to exist on different distributed servers, guaranteeing that the user would have access to the most recent, validated version of that code and supporting parameter data sets.

4 TECHNOLOGY DEMONSTRATIONS

Technology options in the various levels (data, models, presentations, applications, and network) were explored, and demonstration projects were created to show and evaluate their potential. This effort was not just model integration (e.g., connecting the output from one model to the input of another). It constituted package integration, since each applications package usually comes with a data set, a set of models, and an interface to interact with the data and model specifications. The process of integrating packages involves model integration (connecting inputs into outputs with the appropriate control structure), data integration (determining which subset of data to use and how the remaining data get mapped), and presentation integration (determining what inputs are required, how to maintain support, and how to view and navigate the output). The connection adds one new aspect: how to integrate the components separately and then integrate the levels into a new package. Since the packages might be served from different locations, a new aspect of network connection also is involved.

4.1 COMPONENTS

Traditional software packages are custom integrations of models, data, and user interfaces. Sometimes the model is somewhat separated from the user interface. Sometimes the data are stored in a flexible format; other times, they are highly formatted and depend on the model. To ensure a chance of integrating software packages, it is important to separate these components. (Existing software packages usually cannot be integrated directly, and they must at least be separated into components.) These separated components can then be integrated (i.e., models with models, data with data, and user interfaces with user interfaces). Only then can the integrated components be further integrated into a software package (Figure 4).

4.1.1 Models

In the modeling area, the DIAS system offers a conceptual framework for flexibly connecting models. It stresses the separation of the data, model, and interface while maintaining network accessibility transparently. The conceptual model has been implemented in an actual system that was Small-Talk-based and recently migrated to Java. The technology for creating such systems is still developing, and it is quite possible to develop model, data, and presentation components that are independent of this implementation and connected with system tools.

The DIAS concept is to encapsulate models as object methods. The objects can then be quite general, allowing the integrator to choose from appropriate models to provide the method. The integrator can also select how the data needs of various models are translated into an integrated data model and how the user will interact with these data needs. The models can be wrapped in many ways, allowing for different levels of data specification and assumptions (e.g., using full-blown multilevel contaminated zones or simply using a template file and allowing the concentrations to be specified).

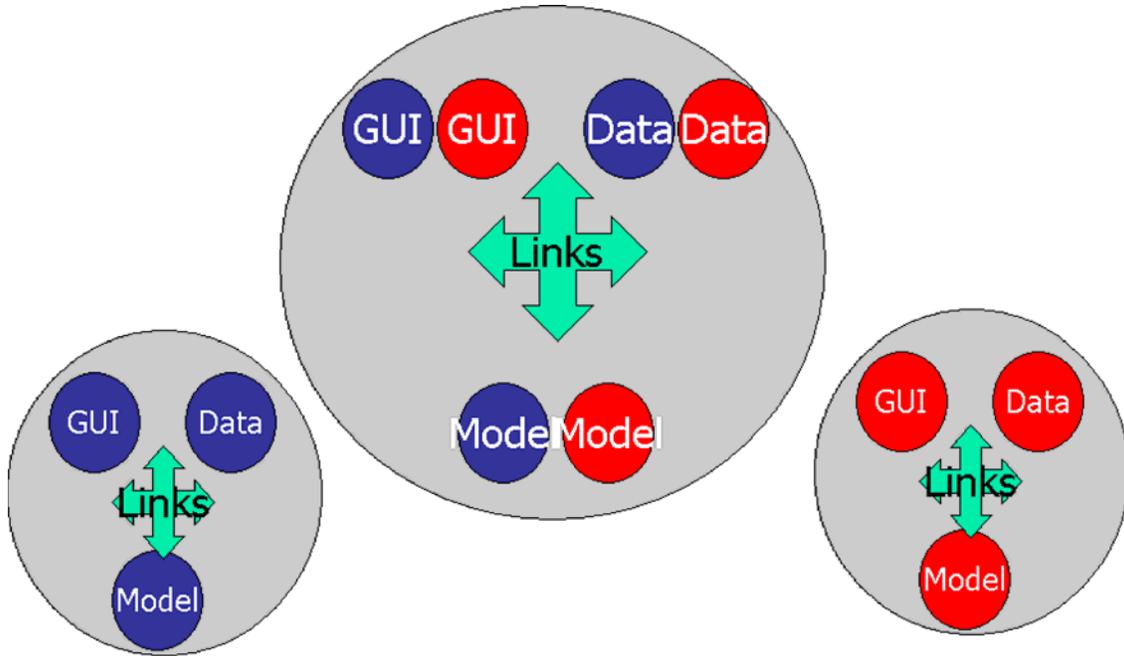


FIGURE 4 Integrating Models, Data, and User Interfaces (Software packages [sides] must be separated into user interfaces [GUIs], data, and model components before integration. The integrated package is formed in a two-step process: integrating similar component types and integrating different component types.)

To use this technique, the data support, modeling methods, and presentation are generally separated at first. Wrappers around the model supply the necessary object methods. The objects are sometimes difficult to choose because of the internal assumptions within the codes. If the original codes are integrated, the options for connecting various models may be few. If the models are more modular, there are more options. For example, in environmental modeling, a time-dependent contaminant and transport media flux (e.g., water) between two physical zones (e.g., unsaturated and saturated zones) is usually sufficient to connect models.

However, often a single, comprehensive model is easy to construct but inefficient to use in a larger modeling system. For example, a generalized external model might require significant computation time. If the exposure geometry is constrained (e.g., contaminated soil with a person standing on top of the soil), the computation time could be greatly reduced by using a more specific model. The specific model is constructed by determining an appropriate way to precalculate some intermediate results that could be used in the constrained model. In this case, it would be useful for an external model to have three methods: a full external model, a limited external model, and a model to prepare the data the necessary for the limited model from the full model.

Allowing models to be encapsulated as general objects enables them to be used by many different integrators to design environments of their own choosing. For example (as described further in Section 4.2.2), a RESRAD object was created and placed on a web server. It was wrapped with a data template file for a typical problem in the Formerly Utilized Sites Remedial

Action Program (FUSRAP). The method required only data on radionuclide concentrations and returned dose versus area in a plot. This same RESRAD object was used with the typical data set and input/output GUI. This example demonstrates the flexibility of allowing different integration techniques (in this case, MS COM, ColdFusion, PopCharts, and ArcIMS).

Standard data communication between objects can be defined, or, through wrapping, modified as needed if sufficient data are provided. Data communication can be local through API calls or across the Internet by using XML. Some HTTP protocols exist to make this communication transparent. MS web services allow an easy transition from a locally called method through argument passing with the Simple Object Access Protocol (SOAP), which translates arguments and method results into XML for communication through HTTP.

New models should be developed with modularity in mind. For example, the RESRAD-BUILD code was highly functionally decomposed in FORTRAN, allowing for simpler conversion to object methods. The RESRAD code is now being incrementally functionally decomposed and separated. In the meantime, the fluxes of contaminant and transport media are being written and read at interfaces between physical media. An example of a fully object-oriented, simplified RESRAD code was written to demonstrate what it might look like.

4.1.2 Presentation

User interfaces that are not used very frequently but are used to build public confidence or meet regulatory requirements must provide access to assumptions, data support, and data interpretation support to facilitate understanding. Tools used to build data support in the interface include context-specific help, sensitivity analysis, uncertainty analysis with default distributions, data validation, data visibility depending on selected options, data feedback presented in graphs and diagrams, and data feedback that presents a state (e.g., default or site-specific). Many data elements are not totally independent of each other. For example, there are many nuclide-dependent parameters. Some codes have options for users to select from many different sources of default values (e.g., state-dependent).

Constructing user interfaces that meet many criteria is difficult and has been known to consume a large fraction of the effort expended to model codes. Even when current-generation languages such as VB are used, the proper construction of customized interfaces can be tedious and error-prone. Some developers have opted for a generic data grid approach to user interaction. This is useful for certain circumstances where variables are not strongly interdependent. Interdependence can arise from various options, selection of details, and validation techniques. This grid interface approach does not allow multiple pathways for specifying data, as do RESRAD-BUILD's and MILDOS's GIS-type systems.

The goal of the interface component development was to automatically support a variety of features yet maintain the flexibility to support custom layout, visual feedback of data, and data dependencies. This goal was accomplished by designing a small set of interface control components that could be easily attached to a database. The output results from a code should be readily available in both textual and graphic forms and be available for use in other codes.

Commercial tools that can be customized to support visualizations such as graphs and GIS displays are available.

The interface components were demonstrated in the RESRAD-OFFSITE-DUST connection for the DUST portion of the user interface using MS VB controls. The same concept was applied to an Internet-tag dynamic server system in the ColdFusion implementation of the MARSSIM application for RESRAD.

The output connections are demonstrated in the MARSSIM connection to POP-Charts and the RESRAD connection to ArcIMS.

4.1.3 Data

There must be a set of data to support the calculation and interface. It should include the user values, default values, upper and lower bounds, and distributions for uncertainty analysis. The data passed to the actual model might be the default data if the model does not allow user input, or the data might be constructed from other input if the model depends on other models' inputs. For example, in the RESRAD-DUST connection, the data requirements to estimate radionuclide migration for RESRAD are precipitation, irrigation, and runoff; the data requirement for DUST is Darcy velocity.

To pass these data, XML can be used transparently through SOAP. The models can also be wrapped so that only part of the data is exposed for the model and user specification. Different data sources must be tracked; for example, an independent DCF library might be created, and special attention might be paid to reassert various default values (i.e., the current value must be kept, and sometimes the state of a default object must be kept).

These data techniques are demonstrated in the databases constructed for the DUST code and the partial database used for the MARSSIM demonstration. The effort required is greater for RESRAD, since the input data have traditionally been stored in flat NAMELIST files. The wrapping translates the database to the NAMELIST file to maintain the code without rewriting. However, the database affects the ease of maintaining functionality and constructing user interfaces.

The output should also be kept in the database so it can be displayed to users in textual or graphical form or be used further in another model. This condition was demonstrated in the RESRAD and RESRAD-BUILD uncertainty analysis database, where the flat files were read to construct a database of input and results that was nearly self-descriptive.

4.1.4 Network

There are many ways to connect users or give them access to web-enabled applications; ASP, CF, and CGI are just a few examples. These techniques can also use some of the interface techniques described earlier. When there is good separation of the interface, data, and models,

the delivery mechanisms can be quite flexible, as demonstrated by the RESRAD/MARSSIM example. The RESRAD code is web-enabled and collects input from a browser constructed from a database that specifies the input and generates an output graph.

Although there are many standards (e.g., RMI, CORBA, DCOM), new technologies are still being introduced to overcome various limitations. These web services run over HTTP protocol, allowing more flexibility and ease in connection. Some integrated development environments (IDEs) allow for easy incorporation of distributed services such as MS Visual Studio.NET. These web services allow programs and databases to easily communicate by using standard protocols like SOAP and XML.

For example, the nuclide service was developed on a .NET service. The user can easily find it and download its API and structures to an application that can use the object and methods much like a local piece of code without having to worry about firewalls. The communication is automatically done in XML.

4.1.5 Integration and Connections

There are many ways to connect models. Wrapping and abstraction to object models were discussed. These model objects may be expressed as libraries that can be easily incorporated into an integration package (e.g., COM or dynamic link library [DLL] objects in MS applications or through web-based ASP/ColdFusion-type languages). Depending on the granularity of the model, there might be great flexibility in connecting it (e.g., the demonstration RESRAD-Object model), or limited flexibility (e.g., DUST model), or in-between flexibility (e.g., RESRAD-OFFSITE with intermediate flux results).

One view would be to have the integrator be responsible for integrating the disparate models, data, and interface components into packages, assisted by the techniques described to ensure compliance with processes, assumptions, and data availability and to generate results that are defensible, understandable, and flexible enough for further processing.

4.2 INTEGRATION DEMONSTRATION PROJECTS

Three demonstration projects were chosen both to address a current need among radiological analysts and to be potentially useful in later applications. The projects demonstrate the wide variety of integration techniques and ways to use components based on existing software packages, new models, and commercial components.

- *Low-level landfill analysis:* Using the NRC's DUST (Sullivan 1993) package and a modified RESRAD-OFFSITE package (Yu et al. 2001), the models were integrated into a desktop application, with DUST providing a leaching source term to the groundwater and RESRAD providing the multipathway dose assessment from that point. The user-interface and model assumptions were maintained.

- *MARSSIM analysis with RESRAD*: The RESRAD model was wrapped with a preprocessor and postprocessor for web execution. The pre- and postprocessors allowed simple connections to a customized, simplified, web-based user interface and commercial visualization graphing and GIS packages.
- *Nuclide web service*: A nuclide data component was developed to allow common access to applications of data structures. The nuclide data were obtained from a distributed server and used by a local application that could then manipulate the nuclide structure in a common technique.

4.2.1 Low-Level Landfill Analysis with DUST and RESRAD

The integration of RESRAD-OFFSITE and the DUST computer code allows users to perform risk analyses of potential releases of low-level radionuclide landfills. The integration was made possible because the RESRAD-OFFSITE model has a feature that allows intermediate contaminant fluxes to be output or serve as input for the remainder of the calculation. In this case, the DUST code was used to generate a modeled release flux at the bottom of the landfill. This flux was then used by RESRAD-OFFSITE for dose risk analysis through the groundwater pathways, including the drinking water pathway and pathways associated with the use of contaminated irrigation water.

To accomplish this integration, the user interface, model interface, and data components had to be separated for each model. Then each component was integrated and packaged in a new application. This practice maintained the data integrity, model assumptions, and ease of use (Figure 4).

The DUST data, model, and interface were separated into three components by designing a database to maintain the metadata of the input parameters. This table included names, units, values, and bounds and could potentially include the grouping and distribution of the data for uncertainty analysis. From these data, RESRAD-OFFSITE was wrapped to allow the user to specify specific configurations of input/output flux planes and the remainder of the input parameters (Figure 5). The output files could be parsed for specific information to be passed to the next software component. The database formed a simple common connection for managing and storing data.

User interface components were developed in both MS Visual Basic and Allaire's ColdFusion. In Visual Basic, the developer could place one of the controls on a form and then associate it with a record in the parameters database table (Figure 6). The appropriate data type and functionality were then automatically packaged in the control. In ColdFusion, the database was likewise used to form data input screens on the basis of a generic database table. Since the user interface is important in communicating model assumptions and facilitating the user's understanding of the model, its development is usually costly. (Sometimes more than half the development time is devoted to custom-developed user interfaces that use standard components. The use of the custom controls and a custom database should enhance the development of user interfaces and reduce the cost of integrating them.)

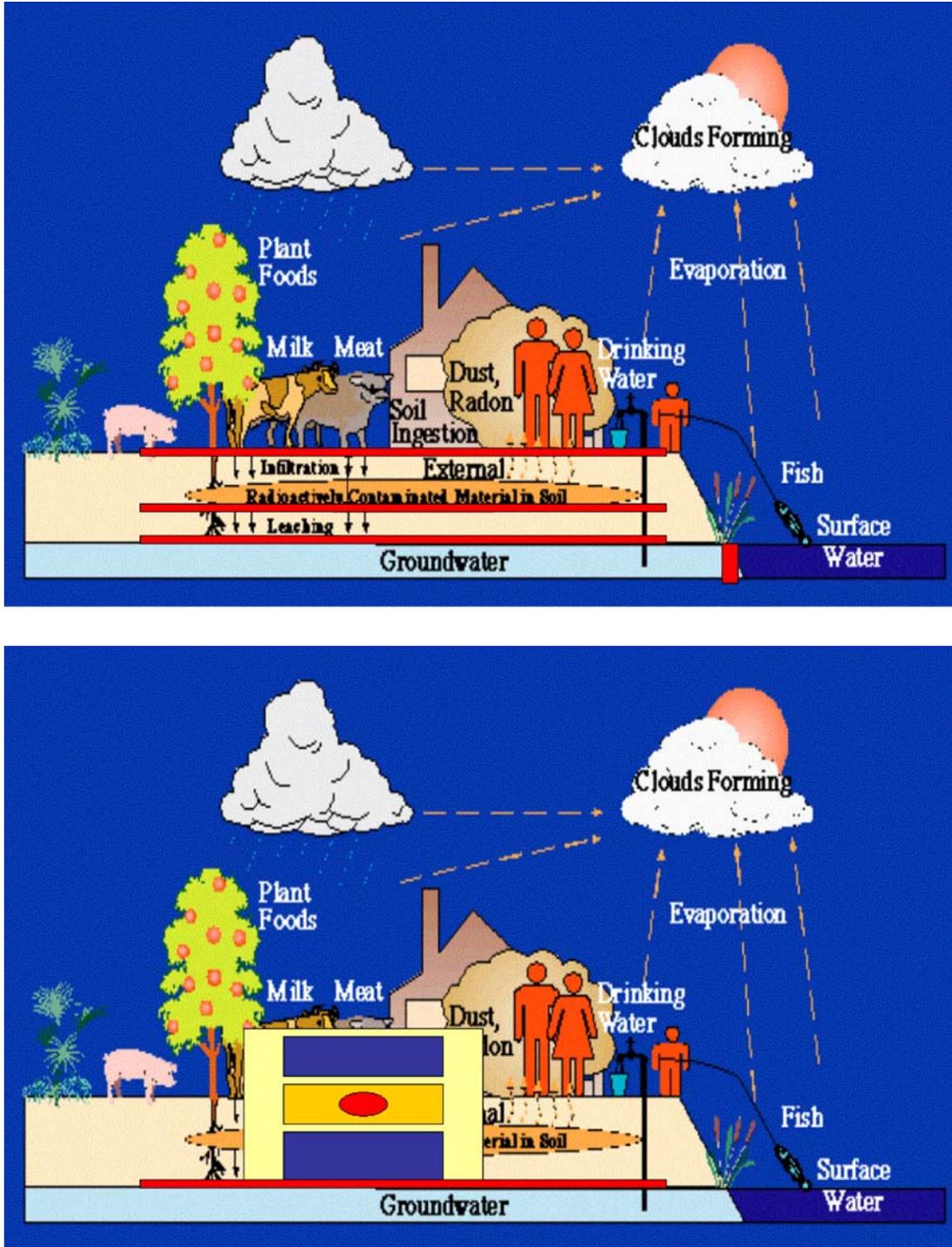


FIGURE 5 Integrating RESRAD-OFFSITE and DUST (The RESRAD-OFFSITE model was modified to accept and output fluxes at certain points in the transport process, including at the groundwater table interface. The DUST results of the fluxes at this location were passed to RESRAD-OFFSITE for further transport and radiological data and risk estimates.)

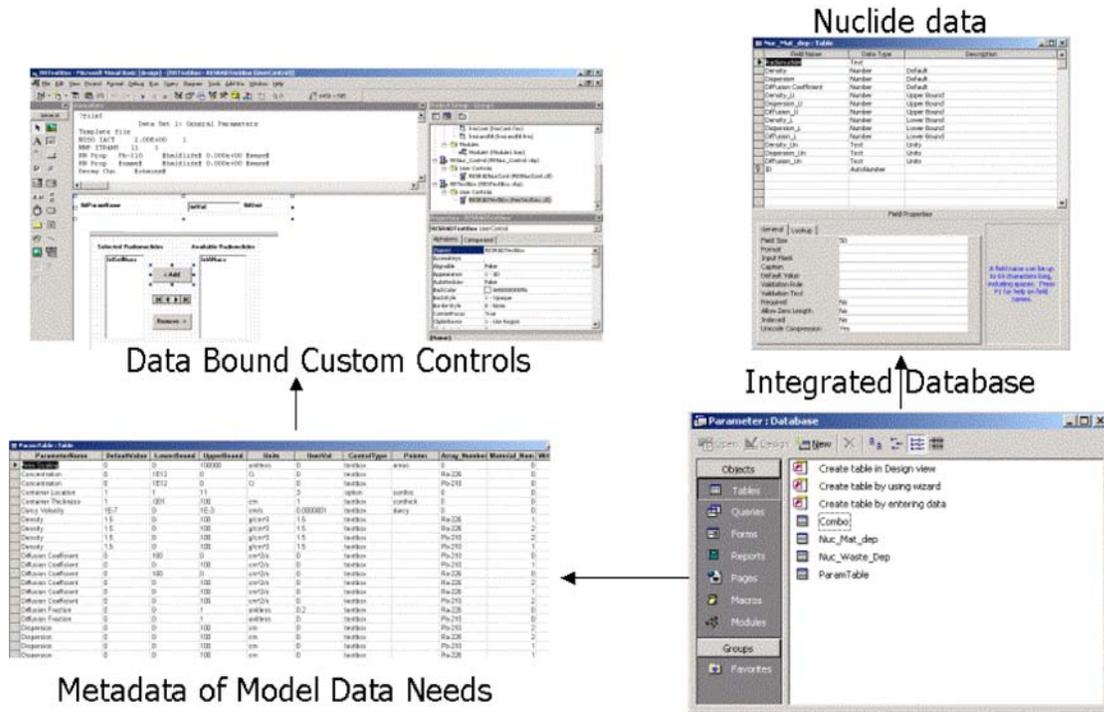


FIGURE 6 Constructing a Custom Interface (The parameter table and nuclide data tables provide a way to quickly construct a custom interface by using a data-bound custom control.)

The user interface can quickly gather data from a user and display them in the database (Figure 7). To wrap the code, the input data must be extracted from the database and passed to the model. Unfortunately, many existing models use a set of formatted input files to pass data to the calculations. This practice makes the conversion from the database to the model difficult, although ideally it only has to be done once to wrap the component. The conversion from the database to the DUST input file was tedious because of the formatting, data array structures, and exceptions. Some other codes like the RESRAD family of codes use the FORTRAN NAMELIST format, which allows flexibly formatted input files somewhat similar to simple XML files (i.e., the parameter name and values are passed without a highly specific format). A full wrapping of the RESRAD or RESRAD-OFFSITE code was not performed; however, in the second project, this technique was used to pass a subset of the RESRAD data through a similar database and onto the wrapped code, which used the database to modify a template NAMELIST input file.

4.2.2 MARSSIM Analysis with RESRAD

MARSSIM is a recent multiagency procedure for finding statistical determinations for radiological cleanup standards. These standards are based on the statistical level of the detector and dose estimates from finite contaminated areas. RESRAD was developed to address cleanup

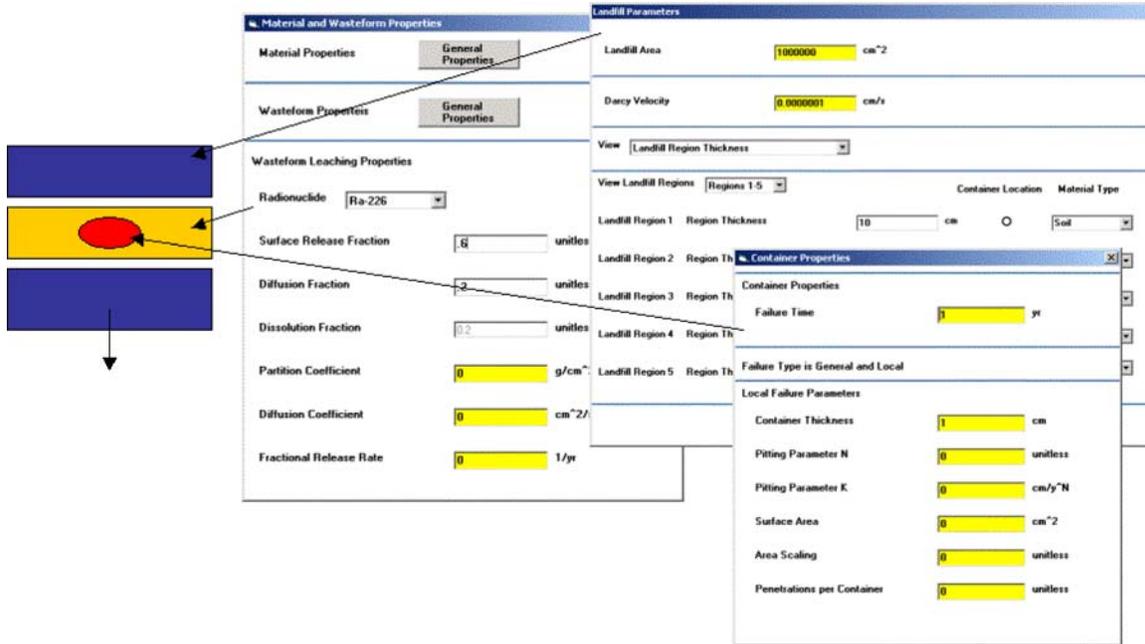


FIGURE 7 Integrating User Interface Forms into RESRAD-OFFSITE Interface (User interface forms are developed for DUST input.)

guideline limits for a set contaminated area. As the contaminated area increases, the dose from the contamination reaches a limit. For smaller areas, the doses are smaller as a result of various scaling factors for the different pathways, such as direct external exposure, food growing activities, dust inhalation, and ingestion. To support MARSSIM activities, RESRAD could be used to generate the dose (or guideline limit) as a function of area. These guidelines could be displayed on a graph and then interpreted on a GIS display of the site with overlain measurements.

While there is a current technique for performing this type of analysis, it is not explained in the RESRAD manual, nor is it a function that the RESRAD interface was designed for. Also, the data visualization of RESRAD was not prepared for easy incorporation into graphs and GIS systems. The purpose of this project is to demonstrate the simple reuse of an existing software application (RESRAD) by wrapping it into an object with a database-driven interface to allow use of custom interface controls. The database interface also allows for easy integration of the model into commercial visualization packages for plotting and GIS. Furthermore, the results can be made accessible on the Internet through a simple web browser interface, giving users easy access to the model, data, and visualizations.

The following steps were taken to wrap RESRAD for this project (Figure 8):

- A template RESRAD input file that contained most of the default input values for the problem was made. The case was set up to include uncertainty analysis of the contaminated area parameter based on a log-uniform distribution. When

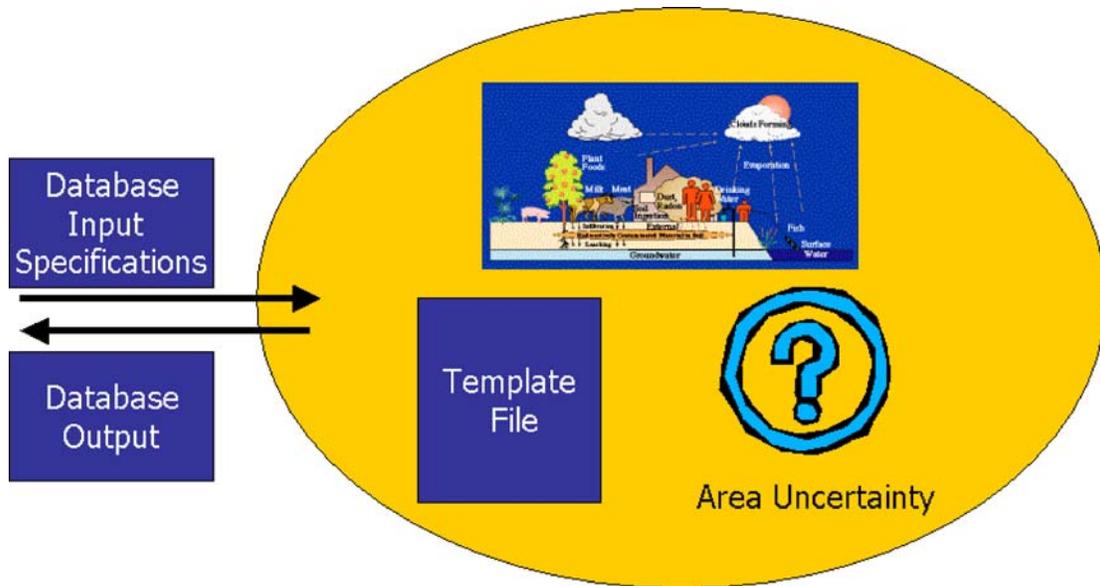


FIGURE 8 Wrapping RESRAD for MARSSIM Analysis (Steps include creating the model DLL, developing a template for the case including an uncertainty analysis, and using a database interface with pre- and postprocessor modules.)

this case was run, the results from the series of RESRAD calculations with different contaminated areas were stored in the standard uncertainty text file.

- A simple FORTRAN DLL to call the RESRAD system batch file was made. This allows the system call on the server to become synchronous (i.e., the call to the DLL waits until the program has finished executing before continuing with further processing).
- A generic Visual Basic preprocessor module was developed to read values from the parameter table in a database and place them in the appropriate places in the RESRAD NAMELIST input file. As mentioned before, NAMELIST input is much simpler to handle than formatted flat files. NAMELIST files are similar to XML files in that they are text-based and associate the parameter name and attributes such as value.
- A generic Visual Basic postprocessor module was developed to read the RESRAD results and insert them into an appropriate database table.

For a code like RESRAD, once these tasks are complete, the resulting modules can be incorporated into a MS COM object for running on the server or made into a VB executable to be called through something like ColdFusion's <CFEXECUTE> statement.

Next an input and output interface must be developed for the system (Figure 9). The interface can be quickly developed through the interface components for ColdFusion. The

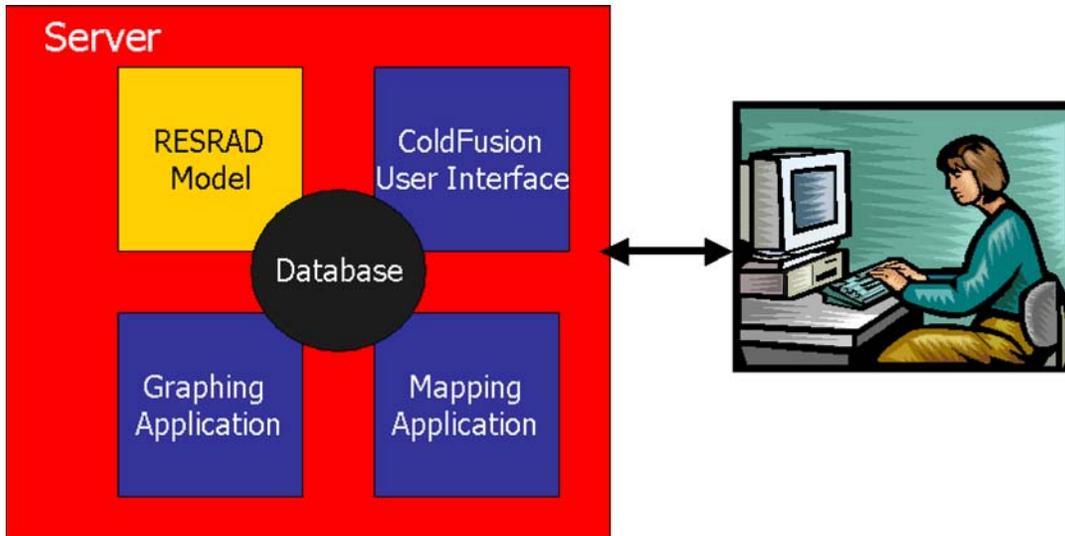


FIGURE 9 Integrating the RESRAD Component to an Interface Input Web Page (developed in ColdFusion) and Two Commercial Visualization Packages (PopCharts and ArcIMS)

ColdFusion template file contains setups for all enabled parameters from the specific database table. The data are collected and validated from a generated HTML page. The user sends the specified input data through the Submit button. On the server, the data are placed in the database, the RESRAD component is executed, and the results are stored in the database.

There are many alternatives for viewing the results. ColdFusion and similar technologies (such as Active Server Pages) have suites of tools for customizing a drilldown report in HTML or in a reporting tool like Crystal Reports. Other commercial tools can generate a more visual display. For this project, the PopCharts Internet graphing package (Corda Technologies, Inc. 2001) and the ArcIMS Internet GIS package were connected to the results database (Figure 10). PopCharts allows results data to be quickly incorporated into a graphing template to show the area factor versus area. The ArcIMS GIS package similarly allows measurement data to be displayed with symbols (type, sizes, colors) on the basis of the measurement level compared to the critical cleanup criteria identified in the analysis. Both of these tools were implemented to generate standard HTML pages. The ArcIMS tool uses extensive JavaScript to enable the user to interact with the map that communicates the data via XML to the mapping server. The measurement data are placed in an acetate layer on top of the map, while the user is still able to interact with the measurements and find out more about them by clicking on them.

This simple demonstration showed how to wrap an existing code, simplify the interface and its construction, and use commercial visualization tools to view all the data integrated on a server. Many extensions could enhance this project: the component could be optimized for performance on a web server; input options could be made available to the user; and the user could be allowed to manage the GIS data more. As is the case for most software development projects, many options are available, but the choice depends on the tradeoffs among data access, complexity, and security involved with a specific project.

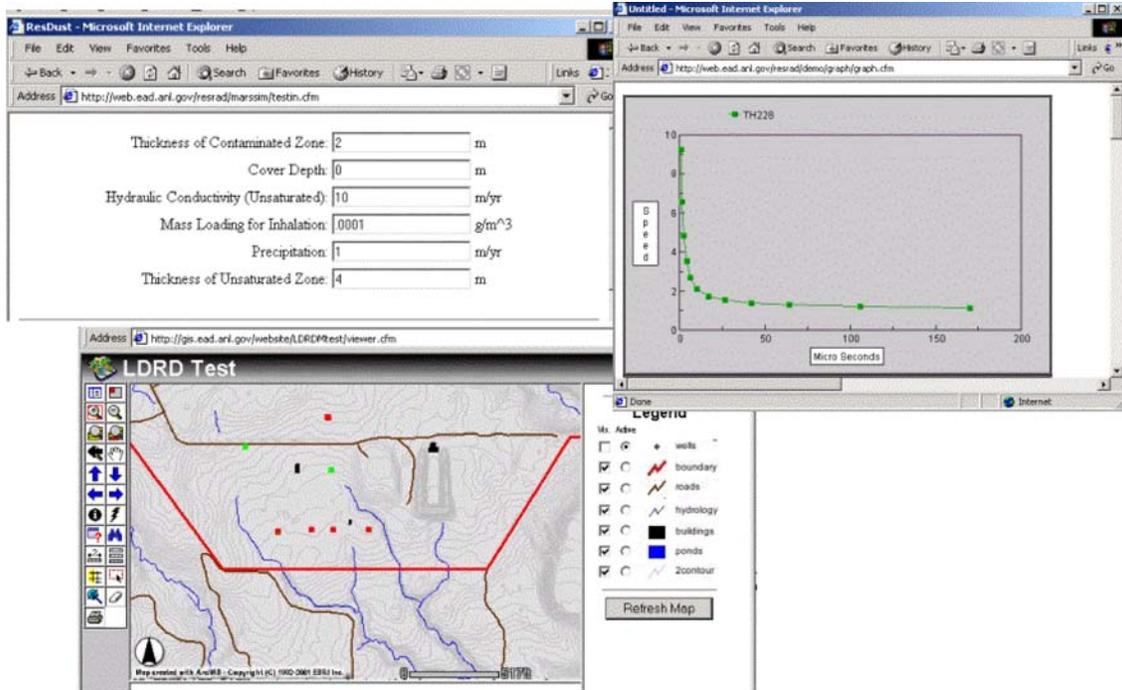


FIGURE 10 Connecting Graphics Packages to Results Database (Input and output web pages for the MARSSIM project show the simplified RESRAD input interface and the results visualization.)

4.2.3 Nuclide Web Service

The previous two projects demonstrated integration on a desktop and integration on a server with web access. Some research modeling environments use components on a set of widely distributed computers. These distributed computing environments allow standard components to be maintained on a few servers that are optimized for performance and maintained with the current versions. Data and models from various sources can be easily connected, and the data can be communicated across the network.

It would be useful to have a set of data sources for contaminant standards or specific data sources from sites that could be integrated and analyzed by means of model and visualization techniques. Tools are currently being developed to facilitate these connections while still maintaining system performance and security. Early connections were difficult to develop and maintain. Even after the connections were constructed, there were frequent disruptions in them resulting from new security techniques such as firewalls (which block communication for standard connection techniques such as DCOM, CORBA, and Java RMI). One example of a new type of tool is the MS .NET web service that is to be released in late 2001. This technique utilizes a standard SOAP protocol to connect components on distributed computers. The communication between the computers is performed by using other standards: XML for the data structure and HTTP for the transmission protocol.

Not only do these services facilitate the use of distributed components, but they also enable simple integration of components. For example, a web service developer can generate a .NET web service by simply constructing an object in the standard MS Visual Studio Integrated Development Environment (VS IDE) by placing a <webMethod> tag in the method to be made available for the service. A test page and site are automatically generated to ensure proper testing and availability of the component.

To use or “consume” this web service, an application integrator can use the same VS IDE on a local development machine. To find out what services are available from a particular server, the VS IDE can simply request that information from the server and present the results, which are integrated into local components available to the integrator. In other words, the web service looks similar to local components. From then on, the development of the integrated package is transparent. Calls are written to the remote object, and the developer no longer has to be too concerned about the computer “handshaking” and passing of data via XML to the server.

Nuclide databases might be a good candidate for a web service. In radiological assessment software, the handling of nuclide data is difficult because of the decay chains and different assumptions about secular equilibrium. Also, some data are quite standard, like the EPA’s Federal Guidance Reports (FGRs 11, 12, and 13). It would be useful to have a web service supply the data in a common form with common functions. Local components could then be generated and shared to handle the nuclides in a common fashion, so that radiological codes could better interact with similar assumptions and handling mechanisms.

A simple web service was set up with a limited set of nuclide data to demonstrate their workings (Figure 11). A method that would take input on a radionuclide and deliver decay chain information was developed. The data were recursively extracted from two database tables. One had the nuclide information (e.g., mass, half-life, dose conversion factors, and potentially distribution coefficients for various media). The second detailed the decay relationship, with primary key fields for the parent nuclide and the progeny nuclide and also a field for the yield (or fraction of the decay that followed that decay path).

These data were automatically inserted into an XML package to be sent back to the requesting computer. Once received, the computer would automatically unpack the XML package, and the data would be ready for use in the local machine. In this project, the data were organized into a linked list of nuclide structures for easy handling in codes. For example, in many radiological codes, the decay process is now handled through many indices that are very difficult to maintain and update. The linked list is simpler to handle and less error-prone, and it facilitates simpler code because indices have been removed and because it can be manipulated with recursive techniques.

These codes were implemented with the Public Beta 1 version of MS .NET. Figure 11 shows the web service method, the testing page from the server, the consumer software, and the consumer interface. Again, not much is shown, because much of the infrastructure that is needed to implement this web service is within the .NET environment, a commercially available

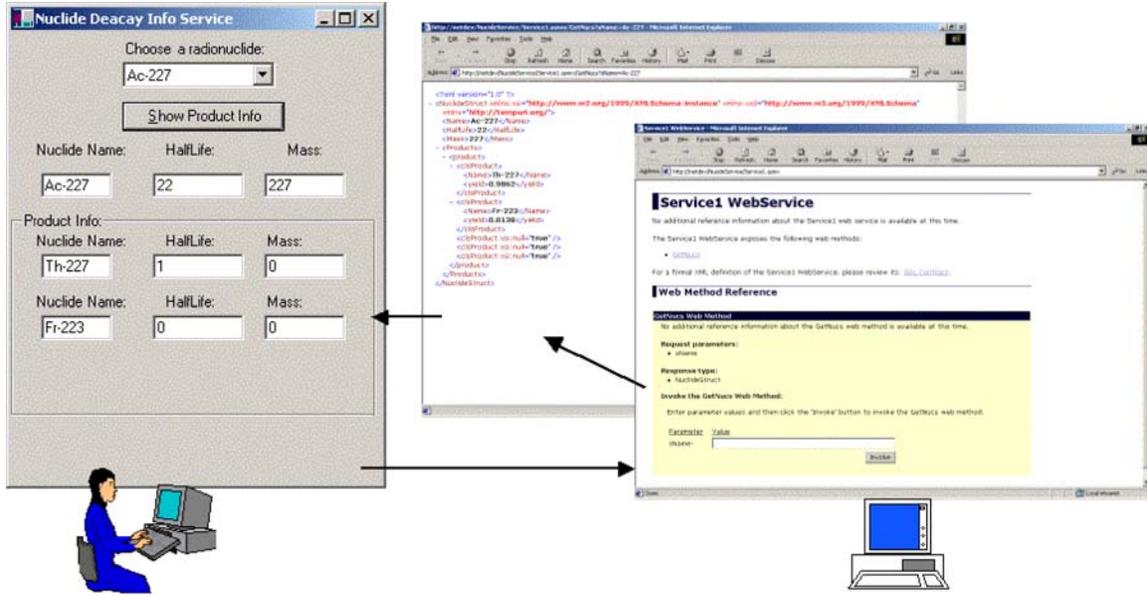


FIGURE 11 Developing and Using a .NET Nuclide Web Service (The object and methods are placed on the server. A web page allows manual checking of the function and demonstrates that XML is used to deliver the results. In the MS VS IDE, the reference to the distributed components can be easily found and inserted. These components are used in the code in a manner similar to that in which the local components are used. The resulting application allows access to distributed data that are used to construct an easily handled linked-list nuclide structure.)

integration package. MS is not the only package available; there are also Java-based packages. It is hoped that through the use and deployment of standards such as SOAP and XML, these web services will be able to interconnect and operate on a variety of servers.

4.3 FUTURE

As new models and packages are developed, some guidelines and standards will help developers design models to be incorporated into larger systems. For example, a simplified object-oriented version of RESRAD's models was constructed (Figure 12). This system will need quite a bit more work before it will be available in this manner for incorporation into a larger system. But the use of components demonstrated here to separate the data, model, and interface has great potential for new models and applications. Before that happens, existing packages can be separated and wrapped in a manner similar to that demonstrated with RESRAD, RESRAD-OFFSITE, and DUST.

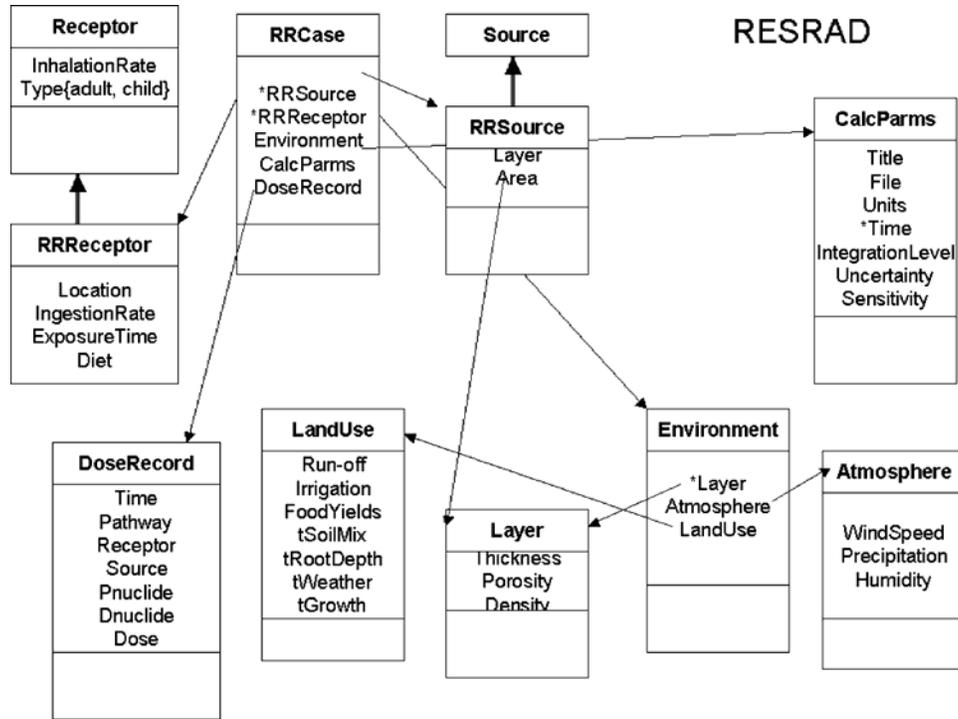


FIGURE 12 Constructing a Simplified Object-Oriented Version of RESRAD (Possibilities exist for creating new models that are object-oriented-based and offer greater potential for component-based systems. A simplified RESRAD model was implemented in an object-oriented fashion. However, much work is required to make a full system with the same data and user interface capabilities.)

5 CONCLUSIONS

5.1 DEMONSTRATION ASSESSMENT VERSUS CRITERIA

The projects demonstrated a small part of the potential for component-based environmental modeling with open integration. Components were developed for the user interface, data handling, model wrapping, connection via desktop, web server, and distributed computing environments. An assessment of this system with regard to the issues mentioned in Section 1 follows here:

- *Flexibility and maintenance:* The system's flexibility was demonstrated when a similar component (RESRAD) was both incorporated in a desktop model integration package and integrated on the web with commercial visualization tools for a different purpose. This integration allows a wide variety of models and end users to be supported. As new components are developed, it is anticipated they will interact so that the applications can be modified to incorporate new model options, data input options, result viewing options, regulatory changes, and tools quickly and efficiently. The open integration environment allows commercial tools to be incorporated rapidly and new Internet options, such as web services, to take advantage of software and hardware (e.g., bandwidth) technologies.
- *Software dissemination:* Barriers to the dissemination and installation of the code package (data, software, and documentation) vary. For desktop systems, some installation packages are initially large but can be reduced for upgrades. Web-based systems require little or no software installation other than a standard web browser. Again, web services also require little or no software installation and also allow local specialized maintenance of the components and data at the distributed sites, so they can be updated without requiring users to download patches. The web-based nature of the download and software package allows web-based user communities to grow, which can generate a critical mass for the use of the component systems.
- *Quality assurance (QA):* The quality of individual model and data components can be assured by the developers. The integrity of the model and data assumptions in the integrated package can be checked by the integrators. Thus, data validity and scenario applicability are left to be determined by the end user. This distributed QA can be more effective, but procedures should be in place to properly document the models so that QA remains intact throughout the process. Development of a large user community would allow discussions and testing of various features and application contexts.
- *Life-cycle development and maintenance costs:* There are always tradeoffs in determining how general a component should be. The flexible system of component integration allows trials to determine the granularity level to which

the components should be specified. For example, some models, such as an external exposure model, might require a generally inefficient model and methods that use preprocessing to make the model more efficient for a more constrained problem. Long-term cost savings might be anticipated from this approach over the traditional development approach. However, the total life-cycle development and maintenance costs will depend on future needs and have yet to be evaluated and assessed.

- *Platform reliance:* As the projects demonstrate, there are many ways to construct, integrate, and execute components. The technologies being developed are becoming more and more open, which means there is less dependence on any one technology. However, as previous component development suggests, there are pitfalls, especially when components are not upgraded and integrated with commercial technologies.
- *Transparency:* Tools can be integrated for both data exploration and facilitating understanding (e.g., sensitivity analysis, uncertainty analysis, and visualization [graphing and GIS]). By incorporating a middle-layer integrator role, the system will present the end user with an integrated package that can preserve the model's assumptions, level of data, and conservatism while still implementing changes in the regulatory process. This type of package will enhance public acceptance and confidence if the model needs to be publicly used or defended.
- *Ease of use:* The flexible integration technique, coupled with the development of interface components and a standard metadata database structure for input parameters, means that user interfaces can be effective (i.e., customized for the needs of the user by being facilitated through the components). Work on the user interface will still be one of the main efforts in model package development, but the use of the components and commercial packages will allow interfaces that will convey an understanding of the model and its results to the variety of stakeholders who are using the environmental modeling system.

5.2 TECHNOLOGY UNCERTAINTIES AND RISKS

Many technology uncertainties and risks surfaced earlier. Many of these issues are addressed by an open system where (1) there is separation of the modeler and the integrator and (2) the modeling and integration tasks can be done with different tools. Such a system also allows a transition pathway that utilizes existing code while concurrently developing new module code. It also allows the integrator to focus on the user's specific need, whether it is for a detailed analysis without a "big picture" understanding or the ability to navigate around issues while applying regulatory requirements to analyze a specific site.

Incorporating many integration techniques also reduces the possibility that environmental models will require a large effort to maintain with the latest technologies and tools. The example of wrapping old FORTRAN codes into objects demonstrates that these techniques allow codes to be used over a wide range of technologies. Using commercial tools and standards allows much of the work to be done by others.

There are some drawbacks, however. Sometimes the technology can be under such rapid development that an integration system might depend on a commercial tool that is supported for only a short amount of time. It is hoped that the components could be developed to be flexible enough so they could be easily incorporated into the new system. A couple of examples of this situation are found in running the FORTRAN code within the RESRAD system. Originally a way was found to run the calculations in the background and still maintain communication with the user interface, which allowed for integrated feedback on the status of the calculations. As the MS operating systems changed from Windows 3.1 to the current Windows version, in almost all upgrades, this code for accomplishing the integration of the model and interface had to be revisited to ensure proper operation. This effort was complicated by the fact that the wide user base meant that many Windows operating systems were being used at the same time, so a way of operating the integration system in not just the new environment, but in all the prior operating systems, had to be found.

Another example was a Java applet written for Internet mapping applications that depended on the server providing the connection to the database. The language rapidly changed, and substantial effort was required to upgrade the original code. In time, the data connection utility was supported and sold only for the new versions of the language (and then dropped, as it was no longer needed in the newer versions). The original applet code was thus somewhat inflexible, depending on the obsolete commercial component, unless effort was made to upgrade the versions of the language.

Also, as technology progresses, changes in application requirements occur not only in response to changes in environmental applications or regulations but also in response to technology changes. Just recently, many requirements have been imposed on government web sites with regard to accessibility to those with disabilities, security, and privacy. However, not all the requirements were imposed with the impending increase in the bandwidth of the Internet in mind. New technologies and techniques based on the connection speeds of the networks will become available.

5.3 ORGANIZATIONAL UNCERTAINTIES AND RISKS

The environmental modeling situation is quite complex because of the range of stakeholders involved in environmental problems. These include government agencies, regulators, end users, model developers, integration developers, and public citizens and organizations.

Government agencies are involved with both the generation and implementation of regulations and with the analysis of internal environmental problems. To meet these needs, the

agencies have funded model and application developments. The agencies have often funded similar modeling efforts for slightly different needs. They hope that there might be a consensus in the model and application development. In the open system, the same model could be used to implement different regulations and workflows. Also, as demonstrated by the use of RESRAD in MARSSIM, analysis models can be wrapped with different interfaces for a variety of uses.

End users must be continually trained in the software to reduce misapplication of the models. With so many components and integration techniques available, end users might be driven into “information overload,” and a critical mass of the users of an application system might not develop, leading to situations where the model might be misapplied. One of the main reasons for the open system, however, is to customize the interface to meet the needs of a particular end user. The Internet could be used to support the user and develop a user community (providing a platform for supplying information about new components and versions, downloading, discussions, and sharing of data). Also, various work-flow processes could be implemented on the web (which has been done for the Environmental Impact Statement Comment Response System) to allow communication among users, modelers, and regulators.

Regulators must be able to determine a model’s applicability, verify data, and gain an understanding of problems through further analyses, such as sensitivity and uncertainty analyses. They are assisted in this process by knowing that end users are provided with a standard user interface equipped with these tools and are given access to standard data. Again, if the system is too flexible, regulators will expend effort investigating the integration of the models, data, and analysis tools instead of the environmental problem itself. This effort could be somewhat alleviated through the QA process for both the constituent models and the integration techniques, especially if those who understood the model assumptions and the interface integrated the models to maintain compatibility. The integrated package could then be subjected to further V&V processes. If a large user community is formed, the package would be scrutinized through quite a few applications.

Model developers risk generating models and components that might not be compatible with other integration systems. A way to work around this problem is to wrap models in a different way to allow integration. However, sometimes the basic assumptions used in the models mean that they cannot be integrated with another set of models. This problem cannot generally be resolved.

Integration developers (integrators) must try to keep the cost of the infrastructure they develop less than the benefit. They must also concern themselves with developing an application that meets the specific needs of the users at the time of development yet has the flexibility for adding new components for new options later. This dual goal involves tradeoffs that can be reduced but never eliminated.

6 RECOMMENDATIONS

Component-based environmental modeling offers many advantages as long as the hazards in developing the system are dealt with. An open system of components and integration techniques offers the hope of addressing issues in an open and shared environment to leverage existing codes in multiple integrations. An open system allows the sharing of models, data, and interface components for many integration techniques (Figure 13).

One such application of this open system would be to integrate the models and data into a flexible system that would be able to deliver information and facilitate understanding over a long period of time. The models and data could be integrated into the decision-making process and field measurements to elicit dynamic feedback on the environmental state of the system, the model, and the supporting data. This feedback loop is depicted in Figure 14 for the long-term stewardship application.

On the basis of the above discussion, the following four recommendations are made:

1. *Develop an interagency consensus on future modeling needs.* It should be recognized that while modularization of the code offers rather attractive and promising features, it still represents a relatively long-term research and development effort. The relevant software technology has begun to emerge only in the past few years and has yet to mature. There is no immediate urgency to commit government resources to a final format or system for development. Rather, it would be prudent to first form an interagency committee to evaluate and agree upon future modeling needs within the context of the currently available or promising software technologies. Given the limited resources of each agency, it is important to reach a consensus, identify priorities, and assess long-term cost ramifications for such an endeavor before a decision is made.
2. *Maximize the use of technologies developed by the software industry.* The commercial software industry has spent billions of dollars in developing improved software technologies in a rather competitive business environment. These technologies are constantly evolving. They are readily available and adaptable for use by the government sector. It is advisable that, given the circumstances, development of model modularization be derived directly from the open market. Caution must be exercised before committing any government resources to similar areas of development, to avoid potential duplication of efforts that have already been made by the industry. Rather, effort should be made in tracking and evaluating the adaptive nature of the emerging technologies for use.

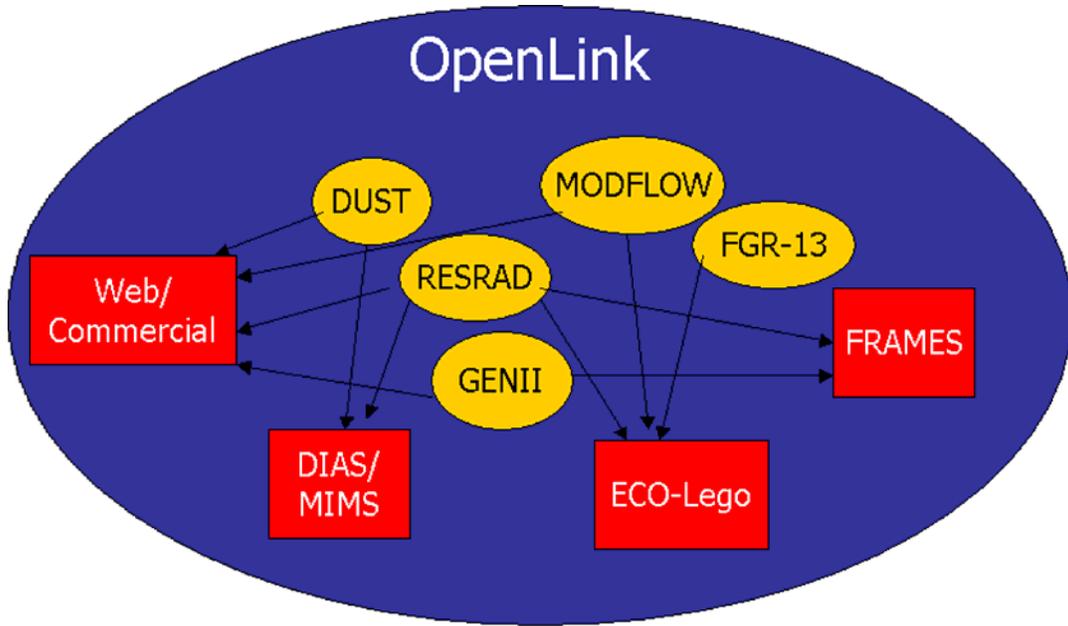


FIGURE 13 OpenLink Sharing of Models, Data, and Interface Components (A flexible and effective system could be developed by sharing models, data, and interface components by following some guidelines and allowing integrators to connect the components into applications for various stakeholders and uses.)

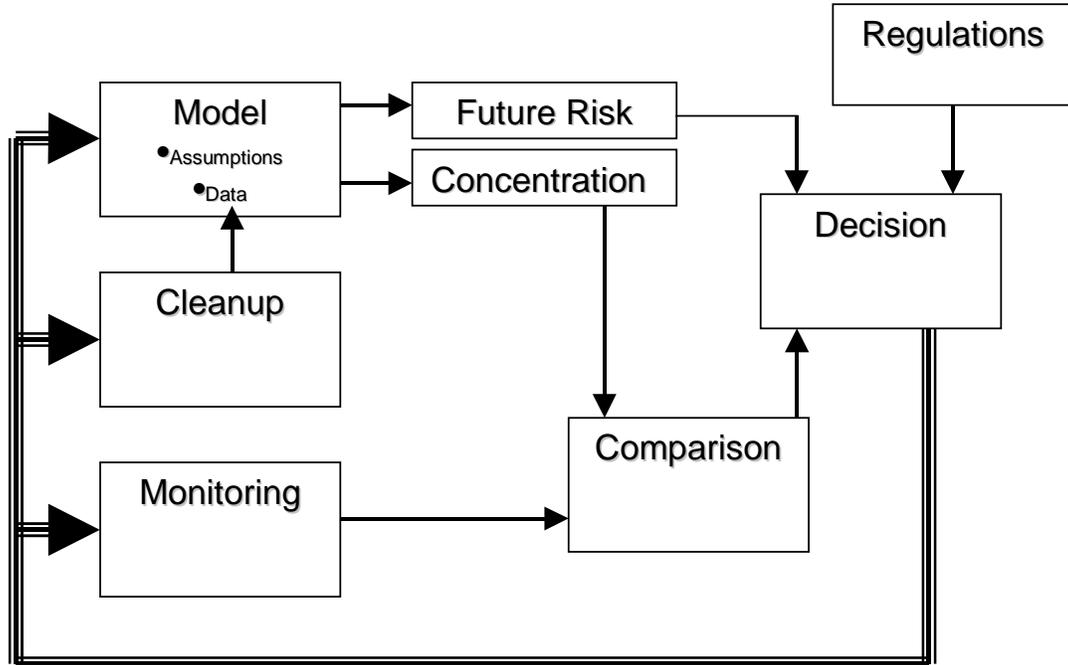


FIGURE 14 OpenLink's Dynamic Feedback System (The OpenLink system could enhance the ability of environmental modeling to assist in the long-term stewardship decision-making process by facilitating an understanding of the design of the cleanup and monitoring activities.)

3. *Maintain the integrity of legacy codes.* While future software development is being pursued, it is paramount that the collective experience and knowledge represented by the tens of hundreds of existing legacy codes be retained and utilized. Since each code has been subjected to various degrees of QA, the already-established integrity must be preserved in the modularization process. The new system should accommodate such preservation.
4. *Minimize dependence on a particular system.* Considering how fast technology in the software industry is evolving, it is important that any system to be developed be very flexible and future-proof. This condition should encompass a rather wide spectrum of parameters, including computer language and architecture. Such an environment would be free of barriers and would virtually remove the dependence on a particular system that needs to be constantly modified or maintained. For this reason, it is advisable to evaluate the current environment for a suitable technology, such as the open-architecture environment discussed above, that may satisfy these requirements.

7 REFERENCES

- Constanza, R., et al., 1993, "Modeling Complex Ecological Economic Systems," *BioScience* 43, Sept.
- Corda Technologies, Inc., 2001, *PopChart Online Documentation*, Lindon, Utah, <http://www.corda.com/support/docs/>, accessed Sept. 2001.
- ESRI, 2001, *ArcIMS 3*, Redlands, Calif., <http://www.esri.com/software/arcims/>, accessed Sept. 2001.
- Forta, B., 1998, *The ColdFusion Web Application Construction Kit*, Que, Indianapolis, Ind.
- Hollis, B., and R. Lhotka, 2001, *VB.NET Programming with the Public Beta*, Wrox Press, Birmingham, U.K., Feb.
- Sullivan, T.M., 1993, *Disposal Unit Source Term (DUST) Data Input Guide*, NUREG/CR-6041, BNL-NUREG-52375, Brookhaven National Laboratory, Upton, N.Y., May.
- Sydelko, P.J., et al., 1999, "A Dynamic Object-Oriented Architecture Approach to Ecosystem Modeling and Simulation," in *Proceedings of the 1999 American Society of Photogrammetry and Remote Sensing (ASPRS) Annual Conference*, Portland, Ore., May 19-21.
- Whelan, G., et al., 1997, *Concepts of a Framework for Risk Analysis in Multimedia Environmental Systems*, Pacific Northwest National Laboratory, Richland, Wash., Oct.
- Whelan, G., and T. Nicholson (editors), 2001, *Proceedings of the Environmental Software Systems Compatibility and Linkage Workshop*, Draft, Pacific Northwest National Laboratory, Richland, Wash., June.
- Yu, C., et al., 2001, *User's Manual for RESRAD Version 6*, ANL/EAD-4, Argonne National Laboratory, Argonne, Ill., July.

