Proceedings of the 2001 Winter Simulation Conference
B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds.

AN OBJECT-ORIENTED PARADIGM FOR SIMULATING POSTAL DISTRIBUTION CENTERS

K. Preston White, Jr.
Brian Barney
Scott Keller
Robert Schwieters
Jacqueline Villasenor

Department of Systems Engineering
University of Virginia
P. O. Box 400747
Charlottesville, VA 22904-4747, U.S.A

William S. Terry
Richard G. Fairbrother
Richard D. Saxton

Distribution Technologies
Lockheed Martin Systems Integration--Owego
1801 State Route 17C
Owego, NY 13827-3998, U.S.A.

ABSTRACT

Discrete-event simulation is an established tool for the design and management of large-scale mail sortation and distribution systems. Because the design of distribution facilities integrates many of the same or functionally similar components, adopting an object-oriented approach to simulation promises significant economies. Instead of coding and verifying models *de novo* for each facility, component subsystem, or individual process, object orientation allows engineers to reuse validated code stored in an objects library. In this paper, we illustrate how the procedural language AutoMod®--a leading commercial simulation package widely accepted in the industry--can be adapted for use within a hierarchical, object-oriented paradigm. A principal contribution is the design of a configuration management plan, which defines a structured process to control and account for the development and maintenance of procedural code and graphics stored in the objects library.

1 INTRODUCTION

Large postal distribution centers process tens of thousands of letters and packages daily. These facilities receive containers of inbound mail from various locations. Mail is automatically inducted into the system and separated into individual pieces spaced along a conveyor segment. Addresses are optically scanned and identified and pieces are then sorted and binned according to outbound destination. The scale of these systems is impressive, with a typical postal distribution facility comprising several miles of conveyors.

Because of the large number and high cost of component hardware and software systems, modern distribution facilities are capital intensive. It is prudent, therefore, that plant managers achieve a high level of flexibility and productivity in their lines. Simulation modeling and analysis help in gaining a better understanding of the dynamic behavior of a system, as pieces of mail flow through from receiving to shipping. This understanding is useful in predicting system performance during the design phase of a project, as well as in aiding subsequent operations management. Simulation allows designers to organize machine layouts virtually (Meyer 1987), improving throughput and cycle times by determining the most efficient arrangement of large systems of interacting people and machines .

Developing simulation code anew for each new facility is both expensive and time-consuming, however, for the very reasons that make simulation the preferred method for analysis. Models of distribution systems are large and complex and generally contain thousands of lines of code. In order for engineers to improve the efficiency and effectiveness of simulation projects, it is desirable to adopt object-oriented approach to model development.

Object-oriented programming resides in the idea of designing and implementing reusable classes of code stored in a software library. This library allows simulation engineers to adapt the work of others within their own simulations, facilitating rapid model development and validation. This approach saves time and money and reduces the likelihood of faulty codes.

In this paper we report on a feasibility study for developing distribution-center software library using the AutoMod simulation language. The study was undertaken by an undergraduate project team at the University of Virginia, supported by professionals at Lockheed Martin Distribution Technologies (LMDT). The main goals for this project were (1) to assess the feasibility of producing object-oriented modules of simulation process logic and graphical representations using AutoMod and (2) to develop a configuration management plan to handle the multiple source and cell files required for this realization. If feasible, the simulation object library will be created and used by LMDT simulation engineers when preparing future distribution system simulations (Meyer et al. 2001).

2 BACKGOUND

Object-oriented libraries are standard in fields that generate large and complex computer codes. Lockheed Martin, already familiar with object-oriented programming in various business units, seeks to capitalize on this approach in its distribution technologies business. Specializing in the integration of large-scale distribution systems, LMDT is the largest supplier of postal automation systems to the United States Postal System and the second largest supplier of material handling equipment worldwide. LMDT designs and manufacturers a large percentage of the machinery it uses in its systems integration projects, including a portfolio of products for address recognition, material sorting, and material handling for postal and commercial customers.

When engineers from different areas of the company simulate the operation of this equipment in distribution lines, a great deal of code with similar functionality is repeatedly written for each new application. A virtual distribution library will enable LMDT to store code as objects and make the object library generally available over the company's local area network. In this way, engineers will be able to reuse existing code within their simulations.

AutoMod is a well-known commercial software package used extensively in the design of manufacturing and distribution systems. In order to create highly accurate models, AutoMod does not explicitly limit the size, level of detail, or complexity of the simulation being developed. In addition, templates available in AutoMod are geared to facilitate the representation and design of material handling systems, including conveyor systems, operators, and automated vehicles.

Perhaps the most salient feature of AutoMod is it's true three-dimensional graphics. Using AutoMod, simulation engineers can create lifelike animations to accompany their code. Navigating these 3-D animations provides an excellent medium to communicate design concepts to clients, in addition to supporting technical studies based on detailed output analyses (Phillips 1998, Rohrer 1999).

AutoMod was chosen as the language for this study because of the strong preference of LMDT's principal postal distribution customer, the United States Postal Service. Because programming in AutoMod is largely procedural, however, the degree to which AutoMod can be adapted to for use within a hierarchical, object-oriented paradigm is unclear. Lockheed Martin seeks to benefit from superior graphical capabilities of AutoMod, while optimizing its coding using object-orientation.

3 EXPERIMENTING WITH PROCESS LOGIC REUSE

Principles of code reuse can be applied to reduce the number of lines of new code generated for a simulation. Process modularization and object-oriented programming are the two main practices explored in this study.

In AutoMod, a complete model may comprise an unlimited number of successively referenced process procedures that are coded in individual source files. However, current practice at LMDT is to develop only two or three source files for each simulation. With so few source files representing the logic involved with an entire distribution facility, implementing even minor changes means delving through a great quantities of code and ultimately requires an intimate knowledge with each aspect of the model. With the current strategy, reuse of code is possible, but problematic. Arrays and sub-functions are means to combat the reuse problem, however these still require traversing multiple levels of logic in order to initiate change.

We developed a simple model as a test case to experiment with modularization in AutoMod. The individual processes chosen for modularization and reuse were an arrival process, a routing delay (conveyor) process, and a service process. These generic processes occur multiple times throughout all distribution facilities and are accepted as the most common building blocks for component level objects.

As shown in Figure 1, the model is a tandem queue with routing delays to the servers. This object comprises five process modules—one arrive module, two conveyor (routing delay) modules, and two service modules. *Loads* (dynamic entities, such as letters or parcels) arrive at the system, one at a time, every five seconds. Immediately upon arrival, loads are inducted onto the first conveyor and delayed while being transported to the first service module.

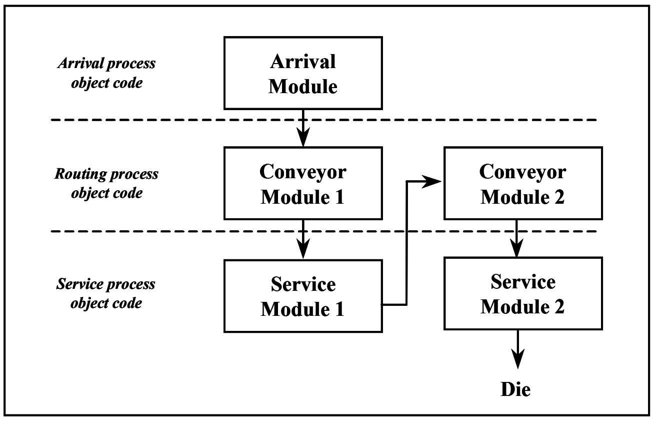

Figure 1: Flow Chart of Process Modules in a Component Level Object

At the first service module, loads wait in queue for a resource with unit capacity and then are delayed by a random processing time distributed exponentially with a mean of 4 sec. After processing at the first service module, loads are inducted onto a second conveyor segment and delayed while being transported to the second service module. At the second service module, loads wait in queue for a resource with unit capacity and then are delayed by a random processing time distributed exponentially with a mean of 3 sec. After processing at the second service module, loads are sent to *die* (depart from the system).

The purpose of the experiment was to assess whether or not source files could be called and recalled independently to create a component level object. To this end, the logic for each type of process module (arrive, conveyor, and service) was coded and stored in one of three separate source files. The three source files were imported into a single AutoMod model file. These files were then edited within the AutoMod process system to define the resources, queues, load attributes, and user-defined variables used in the modules. The length of the routing delays was established implicitly by adding the desired AutoMod conveyer system to the model.

4 MODULAR SOURCE FILES AND INDEXING

AutoMod code for the main module comprises two procedures:

```
/*Initialize routing of loads between
      modules*/
begin P_initV arriving procedure
    set V_chooseproc(1)=P_arrival
    set V_chooseproc(2)=P_conveyor
    set V_chooseproc(3)=P_service
    set V_chooseproc(4)=P_conveyor
    set V_chooseproc(5)=P_service
    set V_chooseproc(6)=die
    send to die
end

/*Route loads*/
begin P_main arriving procedure
    set A_index=A_index+1
    send to V_chooseproc(A_index)
end
```

At the beginning of a run, a single dummy load is created and sent to the procedure *P_initV*. This procedure initializes an array of process pointers *V_chooseproc*, which defines the sequential routing of loads through the five process modules. The dummy load is then sent to *die*.

During the simulation, normal loads are sent to the main procedure *P_main* after completing each process module. In this main procedure, the module routing-index—the load-attribute *A_index*--is incremented by one. The load is then sent to the next process module defined by the routing array.

AutoMod code for the arrival module comprises a single procedure:

```
/*Initialize load indices*/
begin P_initA arriving procedure
    set A_index=0
    set A_servindex=0
    set A_staindex=0
    send to P_main
end
```

Normal loads are created every 5 sec and sent to the procedure *P_initA*. This procedure initializes three load-attribute values, which are used as indices in various modules. Loads are then sent back to the main procedure.

AutoMod code for the conveyor module comprises a single procedure:

```
/*Convey load between sequentially numbered
      stations*/
begin P_conveyor arriving procedure
    set A_staindex=A_staindex+1
    move into conv:sta_(A_staindex)
    set A_staindex=A_staindex+1
    travel to conv:sta_(A_staindex)
    send to P_main
end
```

This reusable module transports a load between any two sequentially numbered stations on the conveyor system *conv*. The stations are indexed by the load attribute *A_staindex*. Loads are then sent back to the main procedure.

AutoMod code for the service model comprises one procedure and one function:

```
/*Queue and process*/
begin P_service arriving procedure
    set A_servindex=A_servindex+1
    move into Q_service(A_servindex)
    get R_service(A_servindex)
    wait for F_servicetime(A_servindex)
    free R_service(A_servindex)
    send to P_main
end

/*Processing time*/
begin F_servicetime function
    if Arg1=1 then return e 4 sec
    else return e 3 sec
end
```

This reusable module inserts the load into the appropriate queue *Q_service*, waiting for the corresponding resource *R_service*, both indexed by the load attribute *A_servindex*. The processing delay is determined from the function *F_servicetime*. Loads are then sent back to the main procedure.

The code described in this section was tested, first by including all of the modules in a single AutoMod source file, then by including the modules each defined in a separate source file. A trace of the simulations demonstrated that processing of loads in both instances of was identical. We conclude from the result that an appropriate scheme for indexing model components does indeed permit a modularization of reusable code segments that can be stored in an library of hierarchical simulation objects.

5 GRAPHICAL REUSE

In addition to the procedural logic files, we constructed 3D graphic files to examine their flexibility within the simulation model. We used two utilities within the AutoMod suite. The ACE utility allowed for the creation of three-dimensional graphical renderings. We built two images that were imported into our AutoMod simulation model as a resource and a queue. The D-Trace utility allowed for the visual enhancement of the conveyor belts, in terms of giving these additional features such as legs, rollers, curbs, and shadows. The finished conveyor illustration was saved as a file and imported into the model.

The two renderings in ACE satisfied our intention of testing the reusability of graphic files. The scanner (or resource) image was only a duplicate of the queue graphic with the addition of a few parts. In this way, we were able to successfully reuse the queue object by adding the necessary components to create an image of a scanner. This method could assist in the creation of large-scale simulations based on the ease of constructing images in a piece-wise fashion.

The D-Trace utility allowed us to enhance the model with regards to the conveyor belt system. We added 3D features such as legs, rollers, and curbs to each conveyor belt to produce a more pleasing image. However, the addition of these aesthetic elements does not support reusability. Each time a change is made to the conveyor belt layout in the *AutoMod* model, the graphical enhancements have to be recreated within D-Trace and again imported into the model.

Both the ACE and D-Trace utilities provide the AutoMod software with great visual capability. The three-dimensional potential of AutoMod is a driving force behind its use. Clients can easily visualize the actual appearance and functionality of their distribution systems. The ACE application is supportive of a reusable methodology whereas D-Trace hinders such a technique.

6 CONFIGURATION MANAGEMENT

With LMDT's goal to minimize simulation development time, the simulation object library will contain an extensive set of processes and graphic renderings to support the development of complete parcel sorting and distribution systems. A large library of continuously updated code can create problems in guaranteeing reliability and repeatability in final products. Without a structured plan to control the development and storage of these simulation objects, engineers developing simulations cannot trust the integrity of the procedural files.

To minimize the integrity problems that can arise in maintaining a preprogrammed object library, a configuration management (CM) plan provides a structured approach to control the development and maintenance of the procedural code and graphics stored in the library. The CM plan outlines the responsibilities of the management team of the simulation object library, the Configuration Control Board (CCB). It also identifies the types of objects stored in the library and the processes for controlling changes to these items.

The CCB plays a critical role in identifying, planning, and controlling changes to the simulation objects of the simulation object library. Its major responsibility is to review all of the change and addition requests to the simulation object library. The other responsibilities of the CCB include evaluating the feasibility and technical adequacy of proposed changes to the library and the CM plan, providing hardware and software engineering impact analyses, reviewing the scheduling impacts of changes to the library code, assigning status audits, reviewing problems with the library and procedures, and providing feedback to users of the library of simulation objects. Figure 2 depicts the review process of the CCB.

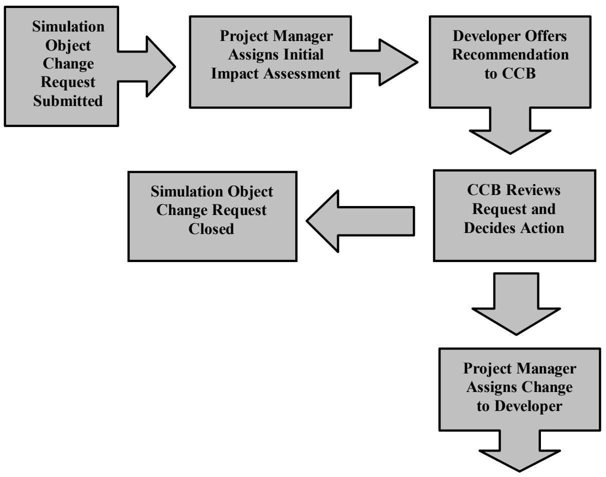
Simulation Object Change Request Submitted
Project Manager Assigns Initial Impact Assessment
Developer Offers Recommendation to CCB
CCB Reviews Request and Decides Action
Simulation Object Change Request Closed
Project Manager Assigns Change to Developer

Figure 2: The CCB Review Process

With a defined management review process, the identification of simulation objects and the enforcement of configuration control procedures ensure the reliability of the changes to objects in the simulation object library. The identification of configuration items sets the foundation for the control of simulation library code development.

The repository for the simulation object library stores the four different object types needed to support the development of simulations. The key configuration items of the simulation object library are objects discussed in the feasibility analysis (the source code files that contain procedural code and the graphic renderings). The repository also stores past simulations for reference and all of the configuration management documentation.

With the configuration items defined, the next task is to introduce procedures for controlling the development environment. Configuration control is the process of managing the changes to configuration items in the simulation object library. The implementation of changes to configuration items approved by the CCB has four major steps in its development cycle to help ensure the reliability of the final simulation objects. To document each step of the code development cycle, four promotion levels are used to identify the status of a file: development, unit test, system test, and simulation. Figure 3 illustrates the development cycle and the promotion schedule.

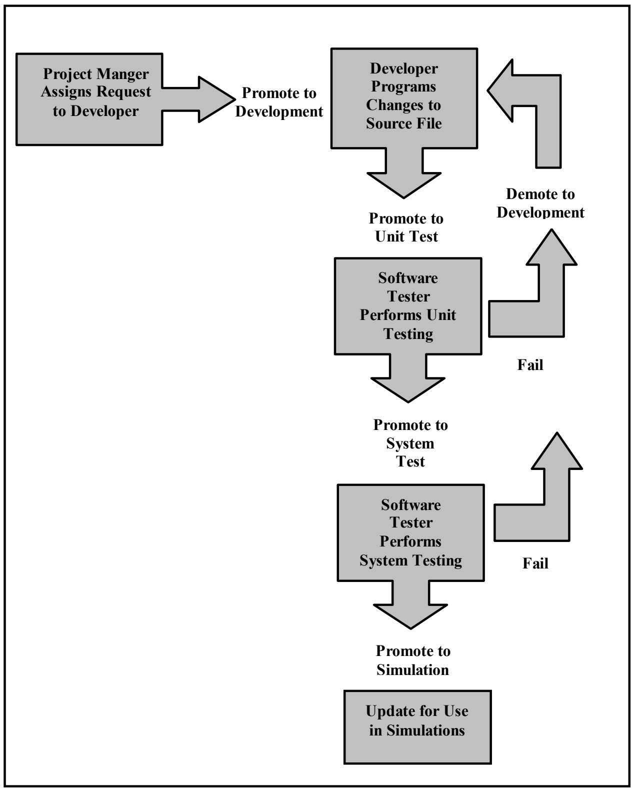
Project Manger Assigns Request to Developer
Promote to Development
Developer Programs Changes to Source File
Demote to Development
Promote to Unit Test
Software Tester Performs Unit Testing
Fail
Promote to System Test
Software Tester Performs System Testing
Fail
Promote to Simulation
Update for Use in Simulations

Figure 3: Simulation Object Development Flow Chart

7 RESULTS

The goals for this project were (1) to assess the feasibility of producing object-oriented modules of simulation process logic and graphical representations using AutoMod and (2) to develop a configuration management plan to handle the multiple source and cell files required for this realization. By means of the example presented in Section 4, we demonstrated that modular source files can be written and integrated into an AutoMod simulation. These source files represent reusable processes that can be stored in an objects library and can be combined into a hierarchy of component subsystem- and system-level objects.

The approach proposed for using these objects is to create a main procedure, which defines a vector for routing loads through the independent modules. The arrival procedure initializes the values of a set of load attributes which index both the current routing step and the number of calls to each module. While the example is simple, generalizing this approach to larger and more complex simulation objects appears to be straightforward. The potential downside to this approach is the large number of load attributes that might be required for a complex routing and the attendant memory requirements.

As outlined in Section 5, using the ACE and D-trace utilities of AutoMod produced the desired reuse of graphical representations. The use of hierarchically structured static sets of objects (for machine parts) generated individual cell files for reuse with the component level objects. Experimentation with D-trace provided a 3-D graphical environment for the entire system. While D-trace does require many redundant steps (a problem with repeatedly exporting and importing of conveyor arrangements) it does offer realism and the attendant advantages of communicating design concepts to clients.

Tying together all the individual source and cell files is the configuration management plan defined in Section 6. This plan gives guidance to managers and engineers on how to execute file changes, sharing restrictions and procedural flow. This plan is essential to providing structure for the object-oriented techniques developed for process logic and graphical reuse.

8 CONCLUSIONS

Based on results of this feasibility study, we recommend that LMDT apply the proposed modularization approach to an industrial strength simulation. Assuming this approach scales, then we recommended that LMDT develop a simulation object library of process and component level objects. Graphical representation of component level objects should also be developed with a reusable methodology. To minimize problems ensuring the integrity of objects in the library, it is essential that LMDT follow a configuration management plan, such as that developed during this project. With a simulation object library of source code and graphical renderings, LMDT engineers will have a tool to economize the development of distribution system simulations.

REFERENCES

Barney, B., Keller, S., Schwieters, R., Villasenor, J., White, K. P., Terry, W. 2001. The feasibility of a multilevel simulation environment tool for designing mail and distribution facilities. In *Proceedings of the 2001 Systems Engineering Capstone Conference*, ed. K. P. White, Jr., and Danner, H.L., 85-90. Charlottesville, VA: University of Virginia.

Meyer, B. 1987. Reusability: The case for object-oriented design. *IEEE Software*, March.

Phillips, T. 1998. AutoMod by AutoSimulations. In *Proceedings of the 1998 Winter Simulation Conference*, ed. D. J. Medeiros, E. F. Watson, J. S. Carson, and M. S. Manivannan, 213-218. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Rohrer, M. 1999. AutoMod Product Suite Tutorial. In *Proceedings of the 1999 Winter Simulation Conference*, ed. P.A. Farrington, H. B. Nembhard, D. TG. Sturrock, and G. W. Evans, 220-226. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

ACKNOWLEDGMENTS

Many thanks to Jerry Banks for a crash course in AutoMod. This paper is a substantially revised and updated version of Barney et al. (2001).

AUTHOR BIOGRAPHIES

K. PRESTON WHITE, JR., is Professor of Systems Engineering at the University of Virginia. He received the B.S.E., M.S., and Ph.D. degrees from Duke University. He has held faculty appointments at Polytechnic University and Carnegie-Mellon University and served as Distinguished Visiting Professor at Newport News Shipbuilding and at SEMATECH. He is U.S. Editor for *International Abstracts in Operations Research* and Associate Editor for *International Journal of Intelligent Automation* and *IEEE Transactions on Electronics Packaging Manufacturing Technology.* He is a member of INFORMS, SCS, and INCOSE and a senior member of IEEE and IIE. He sits on the Advisory Board of VMASC and represents IEEE/SMC on the WSC Board.

BRIAN BARNEY is a fourth-year Systems Engineering student from Rockville Centre, NY. He has been an active volunteer in the Charlottesville community coaching YMCA basketball and tutoring high school students. For this project, Brian experimented with the ACE and D-Trace utilities of AutoMod and focused on the feasibility of graphical reuse. He plans to work in Manhattan, NY, after returning from a trip to Europe in June.

SCOTT KELLER is a fourth-year Systems Engineering Student from Mountainside, NJ. He has volunteered many hours to a local children's hospital and animal shelter. His principal contributions to this project included the experimentation with process logic reuse and correspondence with the client. Scott has accepted a position with Lockheed Martin Distribution Technologies in Owego, NY.

ROBERT SCHWIETERS is a fourth-year Systems Engineering student from Centreville, VA. He competed in the pole vault for the UVA Track and Field team. His principal contributions to this project included the configuration management plan and the graphic renderings for resources in simulations. Robb has accepted a position with Accenture in Reston, VA.

JACQUELINE VILLASENOR is a fourth-year Systems Engineering student from Fairfax, VA. She directly contributed to the creation of the simulation for this project. She plays outside center for the University Women's Rugby team and will compete in her second Nationals tournament this year.

WILLIAM S. TERRY is the Technology Director for Lockheed Martin Distribution Technologies in Owego, NY. He received a B.S. in Electrical Engineering/Computer Science from Clarkson University and a M.S. in Computer Engineering from Syracuse University. He has worked as a system engineer or system architect on a variety of complex system development programs mostly related to supply chain automation and defense systems.

RICHARD G. FAIRBROTHER is Engineer/Scientist with Lockheed Martin Systems Integration, Owego, NY. Mr. Fairbrother has been designing, developing and analyzing simulations for Lockheed Martin Distribution Technologies the last four years. During this period, he has worked on projects for the United States Postal Service, the Royal Mail (UK), AusPost (Australia), as well as a number of commercial material handling efforts. The primary development/analysis tool has been AutoMod/AutoStat, including experience with Taylor ED. Mr. Fairbrother's current focus areas are the creation of a Simulation Development Process for Lockheed Martin Distribution Technologies and the exploitation of AutoMod's ActiveX interface with an emphasis on integrating simulations into a multi-application environment. Additional efforts include the development of simulator/stimulator architectures for the APPS proposal and providing technical support to the 2000-2001 Capstone Project at the University of Virginia.

RICHARD D. SAXTON is Manager, Facilities and Simulation Systems Engineering, Lockheed Martin Systems Integration, Owego, NY. Mr. Saxton manages the simulation engineering department within the Lockheed Martin Distribution Technologies business area. Prior to serving in his current role, Mr. Saxton worked as Systems Engineer and Program Manager in the Distribution Technologies, Avionics, and Systems Solutions business areas at the Owego facility. Mr. Saxton has developed simulations throughout his career beginning in APL and mainframe SimScript environments. His recent efforts include supporting and promoting simulation within the Distribution Technologies business area and the larger Owego site and providing support to the 2000-2001 Capstone Project at the University of Virginia.