Center Directed Research Program

Ecological Modeling System: Conceptual Development and Design Specification

Todd Swannack, James Westervelt, Jeff Hensley, and Robert Kennedy

November 2014

Approved for public release; distribution is unlimited.
The US Army Engineer Research and Development Center (ERDC) solves the nation’s toughest engineering and environmental challenges. ERDC develops innovative solutions in civil and military engineering, geospatial sciences, water resources, and environmental sciences for the Army, the Department of Defense, civilian agencies, and our nation’s public good. Find out more at www.erdc.usace.army.mil.

To search for other technical reports published by ERDC, visit the ERDC online library at http://acwc.sdp.sirsi.net/client/default.
Ecological Modeling System: Conceptual Development and Design Specification

Todd Swannack and Robert Kennedy

*Environmental Laboratory*
U.S. Army Engineer Research and Development Center
3909 Halls Ferry Road
Vicksburg, MS 39180-6199

Jeff Hensley

*Information Technology Laboratory*
U.S. Army Engineer Research and Development Center
3909 Halls Ferry Road
Vicksburg, MS 39180-6199

James Westervelt

*Construction Engineering and Research Laboratory*
U.S. Army Engineer Research and Development Center
2902 Newmark Drive
Champaign, IL 60822

Final report
Approved for public release; distribution is unlimited.
Abstract

As military installations face continued ecological changes caused by changing land use patterns, changing climate, invasive species, and changing missions, the need to forecast the implication of such changes on habitats and at-risk species continues to grow. This requires ecological simulation modeling approaches based on ecological knowledge of the interaction of ecological components at varying time and space scales, at varying ecological hierarchies, and mediated through numerous relationship networks. Because each location and system challenge is unique, a library-based ecological modeling system is required to create cost-effective simulation models. A general purpose, cross-ERDC, widely-adopted ecological modeling system would provide a platform that allows model component reuse, multidisciplinary collaboration, and the ability to rapidly respond to emerging modeling challenges. This reports documents the development of a plan for the creation of an ERDC Ecological Modeling System (EMS) based on the challenges and needs expressed through two ERDC-wide workshops held in 2011.
Contents

Abstract .......................................................................................................................................................... ii

Figures ............................................................................................................................................................. v

Preface ............................................................................................................................................................ vi

Acronyms and Abbreviations ..................................................................................................................... vii

1 Introduction ............................................................................................................................................ 1
   Background .............................................................................................................................................. 1
   Objectives .............................................................................................................................................. 2
   Approach .............................................................................................................................................. 3
   Mode of technology transfer .............................................................................................................. 4

2 Modeling Ecological Systems ............................................................................................................ 5
   Ecological modeling in support of installations .............................................................................. 5
   Approaches to ecological modeling .................................................................................................... 5
   Contrasting modeling paradigms ........................................................................................................... 7
   Problem sets and ecological modeling ................................................................................................. 8
   Modeling and Ecologists ..................................................................................................................... 10
   Component-based modeling ................................................................................................................. 11
   Enhancing ERDC’s ecological modeling capabilities ........................................................................ 11

3 Capability Requirements ..................................................................................................................... 14
   Data management .................................................................................................................................. 14
   Development environment ................................................................................................................... 15
   Component library ............................................................................................................................... 15
   Assembly and execution environment ................................................................................................. 16
   A model execution environment ........................................................................................................... 16
   Visualization and decision support tools ............................................................................................. 17

4 Evaluations ............................................................................................................................................ 18
   Initial system reviews .......................................................................................................................... 18
      Model development systems .......................................................................................................... 18
      Model integration systems .............................................................................................................. 19
   In-depth system evaluations ............................................................................................................... 20
   Evaluation overview ............................................................................................................................. 21
      Model development systems .......................................................................................................... 21
      Model integration systems .............................................................................................................. 22
   Recommendations ............................................................................................................................... 23
   Foundational decisions ....................................................................................................................... 24

5 Development Requirements ............................................................................................................. 26
Figures

Figure 1. Sample Stella Interface .................................................................................................................. 38
Figure 2. ReLogo, NetLogo, and Java code Comparisons. ........................................................................... 54
Figure 3. Zombies in the IDE ......................................................................................................................... 57
Figure 4. Zombies Runtime ............................................................................................................................. 58
Figure 5. Netlogo interface example ............................................................................................................ 74
Figure 6. Part of the catalog of models that come with Netlogo ............................................................... 76
Figure 7. uDig 1.2.2 with Spatial Toolbox open ........................................................................................... 88
Figure 8. AgES UI (integrated OMS3 Runtime), USDA ARS ..................................................................... 90
Figure 9. Components of JGrassTool Aspect .............................................................................................. 92
Figure 10. Cloud computing allows model results on smart phones. ......................................................... 95
Figure 11. Sample FRAMES Simulation Editor ........................................................................................... 99
Figure 12. The FRAMES model integration philosophy (taken from a 2005 PNNL presentation) .......... 100
Preface

This study was conducted for the ERDC Director under the Center Directed Research Program through “The Next-Generation Ecological Modeling System.” The technical monitor was Dr. Pat Deliman.

The work was coordinated by the Wetlands and Coastal Ecology Branch (EE-W) of the Ecosystem Evaluation and Engineering Division (EE), U.S. Army Engineer Research and Development Center – Environmental Laboratory (ERDC-EL). At the time of publication, Dr. Jacob F. Berkowitz was Acting Chief, CEERD-EE-W; Ms. Antisa Webb was Acting Chief, CEERD-EE; and Dr. Patrick Deliman was the Technical Director for Environmental Modeling. The Deputy Director of ERDC-EL was Dr. Jack Davis and the Director was Dr. Elizabeth Fleming.

COL Jeffrey Eckstein was the Commander and Executive Director of ERDC, and Dr. Jeffery P. Holland was the Director.
### Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>3MRA</td>
<td>Multimedia, Multi-pathway, Multi-receptor Exposure and Risk Assessment</td>
</tr>
<tr>
<td>AOI</td>
<td>Area of Interest</td>
</tr>
<tr>
<td>ArcGIS</td>
<td>GIS Software package from ESRI</td>
</tr>
<tr>
<td>ARAMS</td>
<td>Adaptive Risk Assessment Modeling System</td>
</tr>
<tr>
<td>ASTER</td>
<td>Advanced Spaceborne Thermal Emission and Reflection Radiometer</td>
</tr>
<tr>
<td>CCA</td>
<td>Common Component Architecture</td>
</tr>
<tr>
<td>CERL</td>
<td>Construction Engineering Research Laboratory</td>
</tr>
<tr>
<td>CRREL</td>
<td>Cold Regions Research and Experiment Laboratory</td>
</tr>
<tr>
<td>CSDMS</td>
<td>Community Surface Dynamics Modeling System</td>
</tr>
<tr>
<td>DoA</td>
<td>Department of Agriculture</td>
</tr>
<tr>
<td>DoD</td>
<td>Department of Defense</td>
</tr>
<tr>
<td>DoE</td>
<td>Department of Energy</td>
</tr>
<tr>
<td>DVD</td>
<td>Digital video disk</td>
</tr>
<tr>
<td>EL</td>
<td>Environmental Laboratory</td>
</tr>
<tr>
<td>EMS</td>
<td>Ecological Modeling System</td>
</tr>
<tr>
<td>EPA</td>
<td>U.S. Environmental Protection Agency</td>
</tr>
<tr>
<td>ERDAS</td>
<td>A company that makes software for Remote Sensing</td>
</tr>
<tr>
<td>ERDC</td>
<td>U.S. Army Engineer Research and Development Center</td>
</tr>
<tr>
<td>ESRI</td>
<td>A Company that makes GIS software</td>
</tr>
<tr>
<td>FRAMES</td>
<td>Framework for Risk Analysis in Multimedia Environmental Systems</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>GIS</td>
<td>Geographic Information Systems</td>
</tr>
<tr>
<td>GMS</td>
<td>Groundwater Modeling System</td>
</tr>
<tr>
<td>GRASS</td>
<td>Geographic Resource Analysis Support System</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>GRID</td>
<td>A format for saving GIS data in a cell form rather than line form</td>
</tr>
<tr>
<td>ITL</td>
<td>Information Technology Laboratory</td>
</tr>
<tr>
<td>MMS</td>
<td>The Modular Modeling System</td>
</tr>
<tr>
<td>OMS</td>
<td>Object Modeling System</td>
</tr>
<tr>
<td>OpenMI</td>
<td>The Open Model Integration System</td>
</tr>
<tr>
<td>POC</td>
<td>Point of Contact</td>
</tr>
<tr>
<td>SMS</td>
<td>Surfacewater Modeling System</td>
</tr>
<tr>
<td>TOC</td>
<td>Table of Contents</td>
</tr>
<tr>
<td>USGS</td>
<td>U.S. Geological Survey</td>
</tr>
<tr>
<td>WMS</td>
<td>Watershed Modeling System</td>
</tr>
<tr>
<td>XDMF</td>
<td>eXtensible Data Markup Format</td>
</tr>
<tr>
<td>XMDF</td>
<td>eXtensible Markup Data Format</td>
</tr>
</tbody>
</table>
1 Introduction

The Department of Defense (DoD) spends tens of millions of dollars each year on environmental and ecological issues as they relate to or impact two critical missions - training and readiness. DoD lands, which total 30 million acres in the United States, provide habitat for 420 threatened and endangered species, 174 of which are resident on Army installations and facilities (U.S. Department of Defense 2011). The annual cost to the Army for developing and implementing management plans for its 355 installations is $22M. The introduction and establishment of non-native invasive species, often promoted by natural or management induced changes in habitat condition, is a further concern for installation managers requiring considerable efforts to develop and implement effective management strategies (Howard et al. 2003). Similar and equally important challenges confront civil works planners and managers. Invasions of non-native plant and animal species compete with and often displace endemic species, many of which are already at risk. Degraded aquatic and riparian ecosystems fail to meet stakeholder expectations for services (e.g., maintenance of biodiversity, flood attenuation) requiring costly restorative efforts. Concerted effort by Army and the U.S. Army Corps of Engineers (USACE) will be required to meet these difficult challenges.

Key to successfully accomplishing this will be the development of new tools offering capabilities to forecast ecological outcomes associated with the management initiatives. A flexible computational environment that supports the development and integrated application of disparate, multidisciplinary models describing the complex behavior of ecological systems and provides installation and natural resource managers with tools to develop and test management scenarios is urgently needed.

Background

While successful development and application of numerical models describing the movement of water and material across terrestrial, riparian and aquatic landscapes is the hallmark of the US Army Engineer Research and Development Center’s (ERDC) tools to support environmental management. However, there is growing recognition within ERDC, and across the broader ecological community, that ecological questions are complex, interdisciplinary and incompletely answered only using those
approaches. In response, ERDC initiated a series of self-examining workshops as a means to ensure continued leadership in capability development. District and Division participants in a fact-finding workshop (Thornton 2008) identified the need within the USACE planning community for models and tools that allow a more holistic approach to ecosystem management and restoration, as well as modeling frameworks that more effectively integrate a diversity of modeling tools and knowledge bases. Recognizing the urgency for addressing these needs, ERDC’s Environmental Laboratory commissioned an external panel of environmental experts to assess current approaches to problem solving and to make recommendations for future directions (Gunderson et al., 2008). Among the panel’s recommendations were: engaging a broader community in the modeling process; embracing open source modeling; and integrating analyses across multiple disciplines. These recommendations were reinforced by a recent ERDC-wide workshop focused on advancing ERDC’s ecological modeling capabilities (Kennedy et al., 2010).

These efforts underscored the importance to ERDC and USACE of pursuing development of a computational environment that facilitates the effective development and application of multidisciplinary models in support of ecological and environmental management at military installations and civil works projects. The Ecological Modeling System (EMS) proposed in this Center-Directed Research Project (CDRP) addresses two fundamental requirements for meeting this need: the ability to design and develop ecological models and the ability to link those models together or with other environmental models, including ERDC’s hydrologic and material-transport codes. Once developed and implemented, the EMS will promote greater integration across ERDC’s environmental sciences and forecasting capabilities, and enhance ERDC’s ability to effectively address environmental challenges faced by managers of both military installations and civil works projects.

Objectives

The primary objectives of this effort were to:

- Identify current and anticipated future ERDC ecological modeling needs
- Establish requirements for an ERDC EMS to support these needs
- Evaluate existing systems with respect to these requirements
- Develop design specifications for an ERDC EMS
The resulting requirements and specifications will then inform the identification and/or development of a future ERDC ecological modeling system with collaboration across all interested ERDC laboratories.

**Approach**

There are two fundamentally different approaches that could have been taken in this endeavor: technology provider led or end-user led. In other words, software-engineer led or ecologist-led. We chose, in this initial effort, to ask the ERDC ecologist community to identify the characteristics of an ideal EMS that would attract ecologists to capture their understandings of ecological systems in software. These answers can then inform a software-engineer led effort to establish the system with the required characteristics.

The approach involved conducting two ERDC-wide workshops as a means to 1) assess needs and to identify desired capabilities, 2) perform preliminary assessments of existing software systems, 3) assess those determined to offer the greatest opportunity for application in developing the EMS, and 4) develop design recommendations for the EMS. During the first workshop (Workshop I), ERDC ecologists and environmental scientists were asked to identify future ecological research requirements and the ecological modeling capabilities that would be required to address these needs. This workshop, which was held in Vicksburg on 23 February, 2011, involved participants from the Construction Engineering Laboratory (CERL), the Coastal and Hydraulics Laboratory (CHL), the Cold Regions Research and Engineering Laboratory (CRREL), the Information Technology Laboratory (ITL), and the Environmental Laboratory (EL). Workshop discussions focused on three central questions: 1) what are the challenges of ecological problem solving, 2) what ecological modeling capabilities are required to overcome these challenges, and 3) what currently available software environments are potential candidates for an developing an ERDC EMS.

Project team members then evaluated currently available model development systems and model linking approaches, as identified in Workshop I, with respect to identified capability requirements. These evaluations included reviews of documentation and case studies, discussions with developers and experienced practitioners, and hands-on used of software. Evaluations were based on a uniformly applied set of criteria.

Evaluation results were presented and discussed during the second workshop (Workshop II) held 12-13 July, 2011 in Vicksburg. Participants
were assigned to one of two teams, and challenged to design an EMS based on information provided. Consensus reached after comparing and discussing competing designs form the basis for design specifications documented in this report.

**Mode of technology transfer**

This document, and the workshops upon which it is based, provides the primary technology transfer mechanism for this first phase of the project. It documents results to date and proposes the basis for work to be performed during FY12 and FY13. Subsequent phases of the project will employ additional modes of technology transfer. These include: 1) webinars and hands-on training opportunities to encourage the development of ecological modeling capabilities and skills among ERDC researchers; 2) development of an EMS community of practice; 3) development and distribution of the EMS that will allow users to develop, share and link sub-models of components of ecological systems; and 4) technical reports detailing evaluations of the EMS based on case studies (as identified and described in this report).
2 Modeling Ecological Systems

Ecological modeling in support of installations

There are two basic modeling needs to support the sustainability of military installations: managing species of concern and managing vegetation in defense of erosion. Species of concern are at-risk, endangered, or threatened by the loss of one or more environmental factors. Reasons include loss of habitat, fragmentation of habitat, competition from invasive species, loss of natural fire regimes, increased hunting, accidental killing by vehicles on roads, increased predation, and new diseases. Models developed to address the specific salient issues behind one species' challenges are generally not useful to model another species. Instead, a toolbox of modeling capabilities is needed that includes statistical analysis, GIS, predator-prey, metapopulation, competition, agent modeling, chemical interactions with organ systems, among other tools.

Therefore, ERDC must embrace a wide variety of ecological modeling approaches that include a toolbox for addressing the challenges facing military installations and models that augment hydrocodes to address common ecological needs in water systems.

Approaches to ecological modeling

Ecological systems are inherently complex with processes operating at multiple temporal and spatial scales, each influenced by multiple human and natural drivers. Historically, there is has not been a central theme or approach used to develop ecological models; instead these models reflect disciplinary origin and differing modeling philosophies, and as a consequence, are difficult to apply in an integrated and meaningful way. Ideal models are representations of the most salient aspects of reality as perceived by the observer. Depending on the issue or subject to be addressed, the observer might focus on biochemical pathways, organ systems, spread of disease, habitat loss or fragmentation, invasive species, urban development, temperature stresses, predation, inter-specific dependencies, or response to a pollutant. The temporal scale at which these phenomena are observed can vary widely, from seconds (e.g., in biochemical systems) to decades (e.g., response of trees to disturbances like flooding).
A critical impediment to understanding and modeling is the fact that ecological systems are irreducible, thereby obviating the opportunity to conduct simple experiments as a means to reveal first-principle relationships that would allow generalization across differing ecological systems (Jørgensen 1999). Because of this, a large number of ecological modeling approaches have been pursued (see Jørgensen et al. 2011). Perhaps the most popular approach among ecologists is statistical modeling, which is used to present and correlate field measurements. System dynamics modeling is growing in popularity. Such models capture understandings of how systems work. One example is meta-population modeling, which characterizes the dynamics of discrete populations (including birth, development, reproduction, and death) and movements of individuals between these populations. An interconnected set of populations is the meta-population. Agent-based models capture knowledge of how individuals in a system behave over space and time. Selected behaviors of a population are inferred from the collective behavior of many individuals, even though the dynamics of the population are not explicitly defined. Dynamic forestry models forecast the growth of trees in response to climate, weather, soil, groundwater, and competition conditions. This diversity of modeling approaches complicates communicating the results of modeling work to natural resource managers, as there is no standardized methodology for presenting modeling results.

Currently, ecological modeling efforts across the U.S. Army Engineer Research and Development Center (ERDC) are developed in an ad-hoc manner using various programming languages, or using modeling frameworks developed by other agencies, academia or the private sector. There are significant benefits to this approach. Modelers are not limited to an established approach that may not be optimal or even adequate for capturing the salient aspects of the system being analyzed or studied. In many instances this increases efficiency by freeing modelers to use a familiar modeling language, system, theory or approach. Across ERDC’s recent R&D programs, ecological modeling has proceeded successfully in this manner. However, there are some serious drawbacks to this approach that are, in the long run, perhaps reducing effectiveness and efficiencies in the development of ecological models. These include:

- Difficulty linking pre-developed ecological, hydrological, and other models
- Difficulty in linking ecological models developed using differing methods and approaches
• Slow execution of linked models
• Limited reuse of ecological models or model components when developing new models
• Limited sharing of modeling solutions among researchers
• Reduced opportunities to teamwork across researchers, branches, divisions, and labs.

Needed is a corporate solution that facilitates the development and application of ecological models, recognizes and embraces the need for and benefits of diverse modeling approaches and methods, provides the mechanism for more effective ERDC-wide collaboration in environmental problem solving, and hastens knowledge gain by ecologists and environmental scientists. Described in this report are results of initial efforts to design, develop, and evaluate an ERDC Ecological Modeling System (EMS) for meeting these expectations.

Contrasting modeling paradigms

A workshop held at ERDC-Vicksburg in April 2010 provided an opportunity for ecological and environmental researchers from across the ERDC to identify current methods and models, discuss their sufficiency in meeting current and future ecological and ecosystem problem solving challenges, and explore required technical and scientific developments or innovations (Kennedy 2010). Key conclusions from the workshop included:

• Detailed understanding of ecological and ecosystem dynamics and how ecosystems adapt or persist in the environment severely limits development of generalized ecological models.
• Complex system behaviors (thresholds, alternative stable states, etc) and emergent properties (succession, biodiversity, trophic dynamics, etc) present significant modeling challenges.
• Models for hydraulics, hydrology and transport phenomena (e.g., water quality and sediment transport), while providing useful management and decision-making information, offer limited capabilities for overcoming these challenges.

These limitations arise from inherent differences between physical or chemical systems and biotic or ecological systems, the different methods used for studying such systems, and the differing approaches followed when developing models to describe their behavior. Physical and chemical systems are characterized by well-understood, linear interactions between
components and the tendency to progress toward steady state. Because of this, models of such systems have a fixed structure and high generality. These characteristics account for their great success in addressing many water resource related issues.

In contrast to physical systems, ecological systems exhibit unique characteristics that make model development difficult. Ecological systems are structurally dynamic, holistic, self-organizing entities that are organized hierarchically, making model development following physics-based approaches unrealistic. A further challenge for model developers is the fact that the dynamics of ecological systems, which can exist locally in non-steady-state, are often nonlinear and strongly regulated by positive feedbacks (Straskraba 1979). Because of this, Breckling and Müller (1994) suggest that the dynamics of ecological systems can’t be modeled effectively using traditional compartment models, as applied more successfully to hydrology, material transport, and water quality.

Problem sets and ecological modeling

ERDC ecologists and environmental scientists should develop solutions for a wide range of environmental problems, including: degraded ecosystems; invasive species; ecosystem change; the future of populations of “species at risk;” managing the increasing number of federal and state listed species; impacts of toxins and chemicals on biological and ecological systems; invasive species; the spread and impact of disease; and stresses due to climate and land use change. Workshop I included an important introductory session intended to identify ecologically-oriented problem sets confronting the USACE, especially at military installations and civil works projects, that could and should be supported by ecological modeling. While framed in differing contexts, ecologically related problem sets were markedly similar for military installations and civil works projects, as were constraints placed on managers and decision-makers. Common problem sets included:

- **Interpreting behaviors of individuals** – Individual motile organisms respond to their physical surroundings and external stimuli of both abiotic and biotic origin. Responses include active movement toward or away from optimal or sub-optimal conditions, respectively; foraging, including interactions between predators and prey; migration, either diel, diurnal or seasonal; and coordinated group movement, such as flocking or schooling. Correctly interpreting such
behaviors, while currently difficult, is an important management consideration since these often influence patterns of distribution, exposure to contamination or disease, and responses to natural or management-induced disturbance or stress.

- **Managing populations and communities** – Ecologists have long understood the hierarchical nature of ecological systems and the progressive emergence of unique properties across that hierarchy. Populations exhibit age-structure, genetic variability, and density dependence, all of which are emergent properties that influence recruitment, population viability, and dispersal. Communities, as coexisting populations, exhibit gradients, diversity, and trophic structure. Successfully meeting the requirement to safeguard species-at-risk, reducing the impacts of human-induced disturbance regimes and assessing toxicological influences on population viability are significant challenges for managers that will require capabilities to describe and forecast the dynamics of populations and communities.

- **Managing the impacts of habitat and landscape change** – Spatial heterogeneity and change in such patterns over time are inherent properties of ecological systems with significant implications for management. Poor management practices degrade physical habitats with potential impacts to resident populations and increased susceptibility to invasion by alien species. Changes in the patterns of distribution of habitat types (or mosaics), as might be expected by land-use change, human encroachment, or climate change, can adversely affect landscape change. The implications of such changes are significant, especially as related to installations and ranges.

- **Managing and restoring ecosystem structure and function** - Ecosystem structure, defined as the suite of physical attributes of the environment and the anticipated or desired biotic inhabitants that characterize a particular ecosystem, has traditionally guided management and restoration efforts, particularly in the case of efforts involving endangered species or species-at-risk. Of equal and increasing importance are functional relationships that lead to ecosystem structure, particularly as related to services provided to society. The challenge for ecologists and environmental scientists is to develop the understanding, and associated models, which will allow managers and decision makers to guide future ecosystem management and restoration efforts more effectively.
A number of constraints affect how these problems are addressed and the degree to which effective solutions can be identified and implemented. Principal among these are limited funding, short decision timeframes, and increasing spatial and temporal scope, especially as related to project planning. In addition, many management decisions have high case- or site-specificity.

Participants in a recent USACE workshop on the environmental modeling needs of CW planners (Thornton 2008) indicated that many of their efforts, particularly in early stages of the planning process, are funded at or below $100K. This, coupled with relatively short study durations, often precludes the development or application of computationally intensive tools. Environmental managers at installations are faced the similar funding and timeframe constraints.

**Modeling and Ecologists**

Despite the ERDC’s long and successful history of development and application of environmental models, particularly hydraulic, hydrologic and water quality models, ecological considerations have been difficult to incorporate effectively. In most cases, ecologists have supported environmental problem solving by conducting a variety of field and laboratory studies designed to better understand particular aspects of ecological entities or systems, compiling and interpreting information from relevant literature or gathered during studies of similar problems, and/or constructing models that are often conceptual or statistical in form. Information so gathered and interpreted is used to support ‘best professional judgment’ or, in the case of such issues as habitat suitability, to define and evaluate possible ecological endpoints based on information describing potential changes in physical attributes (e.g., water depth, inundation frequency, and nutrient concentration) as generated by computational models.

While progress has been made toward more integrative approaches, as in the area of behavior and cognitive ecology or efforts to include estimates of lower trophic-level responses in water quality models, ERDC’s ecological capabilities have been under-utilized. While ecologists value models, even if only conceptual constructs or statistical relationships based on data, not all have the technical abilities to develop code or use established modeling frameworks requiring a level of programming expertise. Others are more capable of developing and applying fairly complex models. But differences
in modeling approach and methodology and the limits to our understanding of ecological systems have tended to limit productive interactions across the broader ERDC environmental modeling community and within the ecological and environmental sciences community. A mechanism to broaden collaborative efforts, and encourage greater and more effective involvement in modeling by ecologists and environmental scientists is needed.

Component-based modeling

Ecologists and environmental scientists engaged in developing and applying models of ecological systems are confronted with significant issues related to scale. As stated above, processes collectively responsible for an observed system dynamic occur at specific, and often different, temporal (e.g., seconds to decades) and spatial scales (e.g., centimeters to kilometers), or at specific levels of biological or ecological organization (e.g., gene to landscape). In addition, the same outcomes or responses are often ascribed to somewhat different processes based on discipline-specific knowledge or the disciplinary approach taken by the investigator or modeler. Because of this, no single model could adequately capture the dynamics of ecological systems under all problem-solving circumstances.

A library-based approach shifts the focus away from the onerous task of developing monolithic, stand-alone models, which are often unsupported by the current state of knowledge and empirical data, to the development of individual components that provide the reusable or modifiable building blocks for constructing system-level models. This approach offers a number of noteworthy advantages, including: developing or modifying components is ‘easier’ than developing a large, multiple-processes model; components can be developed using a variety of programming languages; and new or modified component models can replace or be added to existing components (He et al. 2002). Since individual components retain their identity and are recognizable by subject matter experts and model end users, this approach offers the added advantage of encouraging greater participation in model development across disciplines.

Enhancing ERDC’s ecological modeling capabilities

Participants in Workshop I and II agreed that a library-based modeling approach offers the best means to more fully engage ERDC’s ecologists and environmental scientists in model-related aspects of scientific investigation
and problem solving. The proposed EMS provides a standard framework within which ecologists and environmental scientists can develop, evaluate, and integrate component models that reflect disciplinary and methodological variety, and by so doing, effectively enhance ERDC's current environmental modeling capabilities. Clear advantages to ERDC of this approach include:

- Empowers ecologists and environmental scientists to more effectively engage in model development and application
- Provides a standard framework in which to address issues otherwise handled on a case-by-case basis
- Provides the ability to infer across spatiotemporal scales and embrace the hierarchical nature of ecological systems (e.g., scaling up from individual-level phenomenon to population-level processes)
- Provides the ability to project long-term dynamics of projects affecting ecological systems and have the ability to compare multiple alternative scenarios
- Overcomes current impediments to linking disparate datasets (e.g., economic, hydrological and ecological data)
- Provides a mechanism to store and access ecological data so problems can be solved more quickly and efficiently.
- Enables environmental scientists to better communicate results of modeling efforts to clients through visualizations and other communication strategies

The primary objective of this effort is, therefore, to develop and demonstrate a component-based ecological modeling system for use by ecologists, environmental scientists, and land managers as a means to better evaluate the consequences of proposed management alternatives and for overcoming difficult environmental challenges confronting military and civil works managers and planners. The proposed main aspects of the EMS are:

- A continually growing library of ecosystem model components that, when applied in concert, capture the dynamics of natural systems. These components will meet minimal standards that will allow connectivity among components and communication with existing external models, as appropriate.
- A model assembly environment to allow an ecologist or land manager to peruse the component library, select components, identify and
satisfy component input and initialization requirements, and establish run-time data selection and/or visualization.

- A model component development application programmer interface (API) to support the computational needs of component developers. The API will provide all of the behind-the-scenes code needed to allow components to interact with other components.
- A model execution environment that supports model execution using desktop computers, an array of computers, and/or supercomputers.

Described in the following sections of this report are (1) results of efforts to better identify ERDC-wide requirements for a library-based EMS, (2) results of evaluations of existing model development and integration schemes, (3) specifications for the design of the EMS, and (4) details of planned efforts to evaluate the EMS utilizing two case studies.
3 Capability Requirements

Environmental problems addressed by ERDC’s ecologists and environmental scientists vary widely. Problem sets are often viewed as being unique and commonalities across problem sets can be obscured by the application of discipline-specific methodologies and models. The result is incomplete integration of expertise and reduced economy of effort. To be effective, the EMS should provide capabilities to overcome these limitations. During Workshop I, ERDC scientists and environmental engineers reached the consensus that in order to best meet ERDC’s needs, the EMS should, at a minimum, provide the ability to:

- simulate ecological processes across multiple scales
- forecast the impact of projects on the long-term dynamics of ecological systems and have the ability compare multiple project scenarios
- embrace and represent uncertainty
- link to datasets and legacy models
- store, manage, and access ecological data
- use state-of-the-art visualization techniques and other communication strategies.

With these critical abilities identified, Workshop II participants focused on specific capabilities for the EMS within the five core capability categories described below.

- data management
- development environment
- component library
- assembly and execution environment
- visualization and decision support tools.

Data management

Ecological problem sets are multifaceted and successful solutions involve obtaining, assessing, and assimilating information from multiple and often disparate sources. Because data are frequently derived from discipline-specific approaches, there is often marked incongruence that complicates integrated analysis and model development, evaluation, and application. The EMS should have the ability to receive input from a range of types and
formats, and data transformations should be easily performed. Given that the majority of ecological models are problem-specific, the EMS should have a data repository where data and model components can be easily stored, merged, and retrieved. EMS repository data should be easily classified and metadata should be required before the data or component can be added to the repository. It is critical that the EMS have the ability to transform data, so that model components can seamlessly communicate with each other.

**Development environment**

EMS users will exhibit a broad range of modeling and computational experience or expertise and an equally broad range of application needs, including the opportunity for subject matter experts to better understand processes and phenomena suggested by their experimental or observational data. The EMS should empower all potential users to successfully develop and/or adapt models or model components. Essentially, EMS users will likely be divided into two general groups: experienced modelers, many of whom are experienced code writers and; subject matter experts, many of whom will have little or no modeling or programming experience. Experienced modelers will prefer to work in their native programming language and the EMS should embrace that approach. In order for the EMS to be successful, however, EMS users with limited programming experience should be able to conceptualize and evaluate processes and interactions using visual tools. EMS users should be able to use their own data to parameterize model components.

**Component library**

Although many models or model components describing processes and phenomena associated with ecological events lack the generality of physics-based models (i.e., models based on first-principles), they can provide a guide or basis with which to build or adapt models or model components for similar applications. These include models or model components derived from a specific set of empirical observations or from experimentation. Also useful are standard model structures or classes that can be parameterized across a range of potential applications. The EMS should allow users to search, select, and retrieve “standard” model structures or classes, which can be parameterized for their models. There should be utilities in EMS that allow standards to be applied to legacy models that make them EMS compliant (or vice-versa). The component
library should be refereed to ensure quality control. The component library should be searchable and useable by any user within ERDC.

Assembly and execution environment

Addressing environmental management challenges requires the development and application of models that effectively capture the complex dynamics of natural systems. Such system-level models might be reasonably constructed from models and model components developed by appropriate subject matter experts. Assembly would logically involve collaboration among subject matter experts, stakeholders, and managers. The EMS should facilitate those collaborations and make model assembly a user-friendly experience. EMS users should be able to conceptualize and quantify the nature and function of complex natural systems and peruse a component or model library for appropriate (sub-) models, including ERDC existing models, all from within the EMS graphical user interface (GUI). Given the large differences among scales for most ecological problems, the EMS needs a method to characterize input data that verifies scale and that considers units. Error messages should be intuitive and helpful for users. The EMS should offer portability across operating systems and should automatically exploit available computational capabilities.

A model execution environment

Given the potential range in user experience and technical expertise, model execution, in either a component model or system-level model, should be straightforward and reasonably intuitive across a range of computing platforms. Model execution should be facilitated by a control panel that incorporates typical visualization strategies (e.g., slider bars and user-friendly input panels). Users should be able to easily run different scenarios, capture results, and visualize output in real-time. The EMS should contain the option to start and stop the model while running and include the ability to change parameters mid-simulation. The model execution environment should have standard debugging tools (e.g., breakpoints, etc) and the ability to track the sequence of events. In addition, the EMS should be able to be run headless (without visualizations) with multiple threads and on high performance computing systems. The EMS model execution environment should also have self-documentation features.
Visualization and decision support tools

Models inform a range of users, from the subject matter expert attempting to better understand the dynamics of a natural system and develop appropriate management alternatives, to those who ultimately make management decisions. Information and how it is portrayed will differ depending on the users’ needs. The EMS should provide effective, easy to use and understand utilities, either internally or through linkages to existing environments, to meet this range of user needs. EMS outputs should be customizable and exportable to commonly-used analytical or visualization packages (e.g., SAS, R, Google Earth, ERDC-ITL tools). Given the large amount of uncertainty associated with many ecological systems, the EMS should provide the ability to visualize uncertainty, including results from sensitivity analyses, parameter uncertainty analyses, and data gap assessments.
4 Evaluations

Several model development and model integration solutions (including existing ERDC capabilities) with potential application in the development of the EMS were identified by Workshop I participants based on previous experiences or familiarity with solutions reported in technical literature. Post-workshop research identified several more potential solutions that were added to the list of systems to be considered. Evaluation of these systems involved an initial review followed by in-depth evaluations of those determined to offer the greatest potential as either a model development system or a model integration system.

Initial system reviews

Model development systems

Model development systems were initially evaluated with respect to the following screening considerations (as developed in Workshop I):

- **Target User** – The categories are traditional-programmer and non-programmer. “Traditional programmer” refers to a user with experience in the development of models using traditional programming languages (e.g., Java, Fortran, C/C++, etc). “Non-programmer” refers to a user, often a subject matter expert, with limited or no experience in model development who would require an environment or language developed more for the non-programmer.

- **Lead developer** – The organization leading the development of the system. (There may be other organizations also deeply involved in development.)

- **Component based** – Whether or not the system approaches modeling from a philosophy of connectable components.

- **Source code availability** – Whether or not the software is open-source to all. (A “no” response means that ERDC could only acquire source code under to-be-determined agreements.)

- **HPC compatible** – Whether or not the software currently runs in an HPC environment.

- **OS requirements** – What is the required operating system under which the software runs? (e.g., Windows, Unix, OS-X, and Java) “Java”
indicates that the software runs under any OS on which a Java virtual machine has been installed.

- **Dimensionality** – What modeling dimensions are supported (1-D, 2-D, or 3-D)?
- **Ecological hierarchy** – Indicate at what level or levels of ecological organization for which the system will accommodate process model development.
- **External connections** – What external software packages have been integrated? The connection may be through files or more tightly coupled through software.
- **Documentation** – What is the quality of the documentation, especially for new users?
- **Language** – What language(s) are supported by the system?

The following model development systems were reviewed with respect to these considerations.

<table>
<thead>
<tr>
<th>System</th>
<th>Developer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netlogo</td>
<td>Northwestern University</td>
</tr>
<tr>
<td>Repast-3</td>
<td>Argonne National Lab</td>
</tr>
<tr>
<td>Model Builder</td>
<td>ESRI</td>
</tr>
<tr>
<td>HexSIM</td>
<td>Environmental Protection Agency</td>
</tr>
<tr>
<td>ASCAPE</td>
<td>NuTech</td>
</tr>
<tr>
<td>Eclipse</td>
<td>Eclipse Foundation</td>
</tr>
<tr>
<td>OMS-3</td>
<td>Department of Agriculture</td>
</tr>
<tr>
<td>Ecobas</td>
<td>Kassel University, Germany</td>
</tr>
<tr>
<td>Stella</td>
<td>ISEE Corporation</td>
</tr>
</tbody>
</table>

Review results, as well as system overviews, are documented in the Appendices.

**Model integration systems**

Historically, models have been developed to address the dynamics of specific aspects of natural systems while holding the rest of the system static. The boundaries of modeled systems were established to be in a steady state condition, which formed the model boundary conditions. With improvements in computational capabilities, it has become possible to simultaneously model several aspects of a system. One approach to accomplish this is to rewrite the models in a tightly integrated fashion.
Another is to run the models as they are, but linked. This has the advantage of avoiding a redevelopment of tried-and-true capabilities. Model linking comes with a variety of challenges, including:

- Data definitions – the definitions of information in input and output streams can be different from one model to the next. Differences can be in units, spatial or temporal extent or resolution, or in certainty.
- Data format – the format of the data within computer memory, in input/output streams, and in intermediate files varies substantially.

Various approaches to model integration are reviewed in the Appendix. Some (XMS, XMDF, and XDMF) focus on the formatting of data representing system state within complex file structures. Others (FRAMES, OpenMI, and CCA/CSDMS) add the notion of encapsulating existing models in control wrappers that facilitate the sharing of information among models. The OMS-3 framework adds the ability for models to exchange information while running simultaneously.

Model integration capabilities are reviewed in Appendix B: Initial Model Integration Evaluations.

Model integration systems selected for in-depth evaluation included XMS (the ERDC family of Modeling Systems), Conceptual Model Builder (CMB), XMDF (eXtensible Markup Data Format), XDMF (eXtensible Data Markup Format), CCA/CSDMS (Common Component Architecture / Community Surface-Dynamics Modeling System), FRAMES (Framework for Risk Analysis in Multimedia Environmental Systems), and OpenMI (Open Model Integration).

**In-depth system evaluations**

Listed below are the three model-development and six model-integration systems, as well as an established ERDC strategy for data and model output display that were selected for evaluation. The names and affiliation of individuals performing these evaluations are also listed.
Evaluations involved searches of the pertinent technical literature, reviews of documentation and case studies, and hands-on use of downloadable versions of the selected software. In several cases, evaluators conducted site visits to organizations responsible for initial development or those involved in ongoing development. Evaluators subsequently presented their evaluations during the opening sessions of Workshop II. Summary overviews of this information are presented below.

**Evaluation overview**

Evaluations of the three model development environments and six model integration systems are summarized below in the context of capability requirements established during Workshop I. Readers are directed to the appendices for detailed evaluations of Repast (Appendix C), Netlogo (Appendix D), Object Modeling System (Appendix E), OpenMI (Appendix F), and FRAMES (Appendix G).

**Model development systems**

Among the model development systems, Netlogo offers the best overall capability. It has been proven to make programming personally accessible to users coming from many different backgrounds including the arts and sciences. Since it is Java-based and freely downloadable, it is immediately available to most potential users. Hundreds of sample models are available and the documentation is thorough and accessible. It is spatially explicit with the ability to import/export raster and vector GIS data, agent-based (individual mobile agents and landscape patch agents,) and time-step
oriented. The majority of the code is freely distributable, and the remainder can be made available through agreements with the developer (Northwestern University). Its greatest drawbacks are that it does not directly offer any component-based modeling opportunities and does not readily facilitate linking to legacy models.

Repast was developed at the Department of Energy’s (DoE) Argonne National Laboratory. Earlier versions of Repast facilitated the development of agent-based, spatially explicit simulation models using Java as the primary programming language. To attract a broader development community, Repast developers added the ability to program graphically with the Groovy language (and now Relogo, a Repast version of the Logo language). Repast uses the Eclipse IDE (integrated development environment), which provides a rich and powerful capability for programmers to develop and run Repast models. The learning curve for non-programmers is substantial, which may deter many potential new users. Repast also currently lacks the ability to work with raster GIS data or support gridcell-based landscape dynamics.

OMS-3, a product of the Department of Agriculture, offers the component modeling capability that Netlogo lacks. It supports fine-grained component-based modeling that allows for run-time feedback using multiple processors and networked computers. An active design and development group centered at Colorado State University, but involving multiple government agencies, currently supports it. The OMS-3 philosophy is to develop many, mostly new, fine-grained components that can be mixed and matched as needed to address the modeling needs of specific locations. It also has the capability to support coarse-grained component development by encapsulating or wrapping legacy models. During the OMS review, EMS team members demonstrated the ability to capture a Netlogo model as an OMS-3 component.

**Model integration systems**

While investigative methods and modeling approaches often differ across disciplines, successful environmental problem solving will require robust application of multidisciplinary information. Thus, integrating across ecological models and between ecological models and hydrologic models is a critical requirement for an ERDC EMS. Two basic integration approaches are reflected in the systems reviewed here.
The earliest approaches mirrored geographic information systems, storing data in standard-format files. ERDC’s XMS series of modeling packages (WMS, GMS, and SMS) take this approach. Each is an assemblage of models that have been modified to allow them to read and write standard behind-the-scene data files. ERDC has been developing a next generation of this approach by combining sets of files that collectively represent the state of a system into a single HDF5 file through the XMDF efforts. Similarly, a European effort has taken the same approach and developed the XDMF solution, also using HDF5. FRAMES, developed by DoE’s Pacific Northwest National Laboratory, also embraces the linking-through-files approach by establishing a series of specifically formatted DICtionary files. FRAMES also provide effective end user tools that facilitate GUI-based model linking, parameterization, testing, running, and viewing.

A new paradigm involves the use of finer-grained models (i.e., each smaller in scope) that communicate with one another directly (bypassing files), and running simultaneously and interactively. Sample integration environments include OpenMI, the CSDMS CCA, and OMS. OpenMI and OMS target standard networked computer systems, while CCA/CSDMS provides high performance computing solutions. CCA/CSDMS components operate like virtual computers in that they offer open ports to which other components can connect. Similarly, OpenMI facilitates model-to-model requests for information. In fact, an OpenMI simulation begins with a request for the desired final output. The model component that can provide that should request time-location specific inputs from other model components, which might make requests to other components and so on. The CSDMS is NSF-sponsored and invites collaboration with other researchers to add CCA/CSDMS-compliant model components to a growing library. OpenMI is European-based and is actively seeking collaborations in the United States. It was designed to allow cross-European modelers to integrate their models to meet EU requirements to model water systems across the continent. OMS is based on more than 15 years of component-based modeling experience. It offers the ability to build a suite of components with minimally invasive programming requirements.

**Recommendations**

Workshop II included a design charrette involving two groups of participants, including those who had performed in-depth evaluations (as described above). The groups were challenged to design an EMS based either on existing model development and model integration systems, or
on systems requiring complete or partial development. Resulting independent designs were presented and discussed. The consensus recommendation was for adoption of Netlogo as a model development system and OMS-3 as a model integration system.

**Foundational decisions**

Perhaps the best way to bridge our chasm is to establish efforts at both sides that will lead to the development of a solid future bridge, and to work with others across ERDC to establish ERDC-wide model linking solutions. Three foundational decisions come out of the 2011 workshops that propose to do just that:

**Foundational Decision 1: Modeling by Ecologists.** The ERDC EMS team will promote the use of the Netlogo modeling environment as a prototyping means for non-programmer ecologists to capture their knowledge of system dynamics in software.

From the ecology side of the bridge we at ERDC have a proven opportunity to dramatically enhance the software skills, effectiveness, and potential for tech transferring the results of ERDC ecological R&D. The Netlogo software described in an earlier section has already been established as a core modeling capability at one of the labs, CERL. Acceptance of Netlogo has taken place one researcher at a time without any mandates or requirements from program leads, directors, or supervisors. Researchers with no technical background, along with those with some preexisting programming skills have tried Netlogo and have chose to repeatedly adopt it for multiple projects. Through on-site training and a robust web-sharing environment, there is every reason to believe that ecological researchers at all ERDC sites will find Netlogo the perfect entryway into formal ecological simulation modeling and therefore an entryway onto the bridge toward the EMS.

**Foundational Decision 2: Modeling by Programmers and Eco-modeling Technicians.** The ERDC EMS team will promote a library-based ecological modeling system that allows components of models to be shared even though completed models may not be reusable.

This approaches the bridge from the standpoint of robust software development. Consider that the fundamental design requirement is that the EMS will provide an environment in which ecologists and ecological modeling technicians will be able to rapidly assemble data components,
model components, visualization components, and analysis components to create locally specific ecological models in support of Army and Corps of Engineers ecological system management challenges.

The component based system could be built upon other ERDC software efforts including tools like CMB (Conceptual Model Builder) and PT123 (a particle tracking program that incorporates hydrological data). A library of basic functionality and agent behaviors can be developed in a C++ framework with generality allowing the modification of existing models and creation of new models. The CMB offers a platform independent GUI front end where the user can create the computational domain, input data, and specify model-specific parameters.

**Foundational Decision 3: Collaborate in Cross-ERDC Model Linking Discussions.** The ERDC EMS team will seek to participate in future ERDC-wide efforts to support linking models that build upon an already long history of development, testing, and evaluation.

It is clear that the art and science of modeling systems will continue to evolve toward more and more interdisciplinary systems. Legacy single-discipline systems were originally developed as stand-alone efforts. As the desire, need, and capacity for modeling larger aspects of systems developed, the modeling community responded by finding ways to link these legacy systems with capabilities such as XMS, XMDF, XDMF, CCA/CSDMS, and FRAMES. The result can be seen in a long list of course-grained, unique, fragile, but working multidisciplinary models. The evolutionary trend has now taken developers to tightly integrated fine-grained component-based modeling approaches that allow for more flexible modeling in which components are capable of interacting during simulations. This approach also avoids the generation of temporary files.

It is also clear that a cross-ERDC solution that will allow ERDC modelers to develop multidisciplinary models requires cross-ERDC participation. Therefore the ERDC EMS team is anxious and ready to participate in such an endeavor, but is not sufficiently presumptuous that it can unilaterally establish an accepted solution.
5 Development Requirements

For the ERDC EMS, the previous chapter concludes with key decisions/goals that involve:

- Adoption of Netlogo as a prototyping environment
- Adoption of a library-based modeling approach

This chapter identifies the efforts proposed to create the ERDC EMS, efforts guided by the software needs for completing EMS-based models. These efforts involve developing an ERDC-based EMS that modifies existing ERDC software packages. EMS version 1 will be developed using existing ecological models developed by ERDC scientists ERDC and modifications to ERDC software packages which will facilitate modeling ecosystem processes, data inputs, data outputs, model visualizations, and model documentation. Efforts also include documentation and technical transfer tasks.

Below we briefly describe the models and then identify the EMS components that will be required to create these models.

Proposed EMS development

Twelve representative ecological models were chosen as a basis for EMS development. Each model will be deconstructed into their essential ecological constituents and then reconstructed as EMS component libraries. These models consisted of agent-based models that were used to determine population persistence, at various scales, across a wide range of environmental conditions. Models varied in complexity, ranging from large scale, spatially-explicit, coupled population-landscape dynamics models to spatially-implicit models focusing on population genetics. Models were developed within the Netlogo environment or Visual Basic environments. EMS development will consist of determining similarities among models by determining the common ecological processes represented across the 12 representative models.
Required EMS ecological modeling components

The components that will be needed to support the development of the spatially-explicit population models are outlined below. They are grouped into component categories described in Appendix E: Component Based Modeling. Each component is described and recommendations on how the development of the component might proceed.

**Input components**

Description: Vector GIS map input to support the ingesting of lines, polygons, and points.

This capability may need to be developed by OMS or ERDC, but Java-based libraries already exist that can be exploited. For our demonstrations, this is only needed for display purposes and potentially for initializing the location of individual agents (snakes and birds).

Description: Raster GIS map input to support the initialization of landscape variables used to define the state of system patches.

This capability exists within OMS.

Description: Table ingestion component.

Various model and sub-model initialization parameters are provided in the form of a table.

Sample tables include data to initialize the state of agents and weather information. This component may be separately developed for each table type. ERDC is the likely developer.

**Ecological agent components**

Description: Bird behavior dynamics as discrete agents.

This will capture the locational and breeding behavior of the black-capped vireo. ERDC will develop.

Description: Snake behavior as discrete agents.

This will capture the predatory behavior of the snakes at Fort Hood that prey on the blacked-capped vireo nests. It will be developed by ERDC.
Description: Fort Hood shrub management.
This will initially be a schedule of management actions resulting in the creation, movement, and destruction of brush piles. It might turn into an autonomous agent that makes these decisions based on other things happening in the model, budget constraints, and the timed availability of equipment and people-power. It will be developed by ERDC.

Description: Land temperature calculator.
This is a raster-based agent model representing the landscape. It will calculate the landscape temperature based on land cover, soil characteristics, air temperature, rainfall, time of day, and day of the year. It will be used by the snake behavior component and will be developed by ERDC.

Description: Synthetic weather generator.
This will output weather in the form of min, mean, and max temperatures and rainfall. It will be based on and may encapsulate the synthetic weather generator developed by Aaron Byrd and Aaron Lee through the climate change CDR. It will be developed by ERDC.

Description: Flatwoods Salamander as discrete agents.
This agent-based animal behavior component will capture the dynamics of the amphibian. This will contain all animal dynamics including movement, birth, death, and transformation. This will be developed by ERDC.

Description: Vegetation growth as patch dynamics.
This is a patch-based simulation model of vegetation growth and death dynamics in response to weather and time-of-year. It will be used to potentially accommodate grass, forbs, and shrub growth and account for above- and below-ground biomass.

**Greater system components**

Description: Ground and surface-water dynamics.
This will encapsulate an ERDC ground and surfacewater model (probably GSSHA). This existing model will be encapsulated as an OMS component. ERDC will be the developer.
**Visualization components**

Description: Landscape state visualization.

This run-time system state visualizer will display images representing the changing state of the system over time. It will allow for the overlay of vector and point data. Description: Run-time charts and graphs.

This run-time system state visualizer will allow the modeler to display the state of the system during model runs in the form of charts and graphs. The OMS developers will further develop existing capabilities.

**Data translation components**

Description: Raster simulation translation

Different landscape models may operate at different scales and within different projections and coordinate systems. Existing Java-based libraries will likely be employed. This module may not be needed for the planned EMS demonstrations. Development will be done in collaboration with the OMS developers.

**Output components**

Description: System state capture in tables

This component will allow the modeler to select system state information to be sampled and saved for later analysis and visualization. This general purpose component will be developed in collaboration with (or perhaps by) the OMS development team.

Description: System state output to Corpsmap 3-D

Select system state information will be saved as time-series images for display in ERDC's Corpsmap 3D environment. ERDC will be the developer.

**System state component**

Description: System state manager.

The goal of this component is to provide a single exchange point for system state information to alleviate the confusion of potentially all components communicating with all other components. This component will provide the system state exchange point for all model input, output,
and shared system information. This will be developed by ERDC, but in tight coordination with the OMS development team.

**Data management**

OMS-based spatially explicit dynamic simulation models are all initialized with the state of a system at the beginning of the simulation time. As the state of the system changes, some (or all) information can be saved for later display or analysis. It is often desirable to store all input data together to ensure that simulation runs can be easily repeated with minimal chance of the input data changing. Because of the variety of input data forms, types, and sources; and the manner in which different modelers store and access data, OMS has not established data management requirements.

In the process of developing EMS demonstration models, care will be taken to identify guidelines, approaches, and conventions that can be recommended to future ERDC OMS modelers.

**Component library**

The EMS project will establish an EMS web site that will provide all of the products developed initially by the EMS CDR project and later by ERDC researchers and projects that will be using the EMS. This web site will meet ERDC standards and specifications. Sections of the website will provide the following:

- A splash introduction page that points to all of the sites resources
- OMS/EMS software download for various platforms and versions
- OMS/EMS documentation including manuals, presentations, articles, and technical reports
- An EMS component library that contains software, documentation, metadata, user reviews, and comments.

**Component assembly**

To turn components into a working model, one must combine them together in a meaningful way, associate them with input data, and connect them to visualizers and data output files for later analysis. Through OMS-3, components are connected through relatively short text files that:

- Identify the components that will be used in the model
• Set model parameters
• Connect components
• Set run-time instructions

The OMS-3 graphical user interface allows one to create, open, edit, and run models. This interface has been developed to assist the text-editor oriented model developer. Earlier versions of OMS contained a graphical-based component assembly environment that, through representations of components as graphical icons, allowed model technicians (non-programmers) to select, connect, and parameterize models. Components could be easily connected to input files and to output visualizations and data files. The OMS development team is planning to release such an environment for OMS in the near future.

Model execution

Model execution in OMS can be accomplished via the OMS graphical user interface, or through command-line interfaces and batch files.
References


Fife, M. A. 2006. A study of model integration in conjunction with the extensible model definition format. Brigham Young University.


Appendix A: Initial Evaluation of Model Development Systems

Netlogo

<table>
<thead>
<tr>
<th>Website</th>
<th><a href="http://ccl.northwestern.edu/netlogo">http://ccl.northwestern.edu/netlogo</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Target User</td>
<td>Ecologist</td>
</tr>
<tr>
<td>Lead developer</td>
<td>Northwestern University</td>
</tr>
<tr>
<td>Component Based?</td>
<td>No</td>
</tr>
<tr>
<td>Source code available?</td>
<td>95% open source</td>
</tr>
<tr>
<td>Active development?</td>
<td>Yes</td>
</tr>
<tr>
<td>Agent-based?</td>
<td>Yes</td>
</tr>
<tr>
<td>GIS oriented?</td>
<td>Yes</td>
</tr>
<tr>
<td>Multi-purpose?</td>
<td>Yes</td>
</tr>
<tr>
<td>HPC?</td>
<td>No</td>
</tr>
<tr>
<td>OS</td>
<td>Java</td>
</tr>
<tr>
<td>Dimensions</td>
<td>3-D</td>
</tr>
<tr>
<td>Hierarchy</td>
<td>Landscape; Population; Individual</td>
</tr>
<tr>
<td>External connections</td>
<td>Statistics; GIS; Mathematica</td>
</tr>
<tr>
<td>Documentation</td>
<td>Excellent</td>
</tr>
<tr>
<td>Language</td>
<td>Logo</td>
</tr>
</tbody>
</table>

The Logo language was originally developed in 1967 as an educational tool to introduce children in grade school and beyond to programming concepts. It is a high-level language, meaning that is provides a rather extensive vocabulary using common English words that allows for the development of short, readable, yet powerful programs. Since its inception, nearly 200 implementations of the language have been developed and, one of the latest and most active is Netlogo. The center of Netlogo development is at Northwestern University, but the development and application community is actually world-wide, resulting in the contribution of many extensions. Netlogo is particularly attractive to landscape ecologists that work simultaneously with individuals (i.e. agents) interacting with each other and the landscape. The user interface initially appears deceptively simple; designed to provide easy and successful initial user experiences. Netlogo is
reviewed more deeply in a later section of this document – based substantially on its rapid acceptance among nearly a dozen ERDC researchers beginning in 2008.

### Repast-3

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Target User</td>
<td>Programmer</td>
</tr>
<tr>
<td>Lead developer</td>
<td>Dept of Energy - Argonne Natl Lab</td>
</tr>
<tr>
<td>Component Based?</td>
<td>No</td>
</tr>
<tr>
<td>Source code available?</td>
<td>Yes</td>
</tr>
<tr>
<td>Active development?</td>
<td>Yes</td>
</tr>
<tr>
<td>Agent-based?</td>
<td>Yes</td>
</tr>
<tr>
<td>GIS oriented?</td>
<td>No</td>
</tr>
<tr>
<td>Multi-purpose?</td>
<td>Yes</td>
</tr>
<tr>
<td>HPC?</td>
<td>Yes</td>
</tr>
<tr>
<td>OS</td>
<td>Java</td>
</tr>
<tr>
<td>Dimensions</td>
<td>2-D</td>
</tr>
<tr>
<td>Hierarchy</td>
<td>Ecosystem; Landscape; Population; Individual; Chemical</td>
</tr>
<tr>
<td>External connections</td>
<td>Statistics; Visualization; GIS</td>
</tr>
<tr>
<td>Documentation</td>
<td>Excellent</td>
</tr>
<tr>
<td>Language</td>
<td>Java; Groovy</td>
</tr>
</tbody>
</table>

Repast is the Recursive Porous Agent Simulation Toolkit, developed by a team at the University of Chicago in conjunction with DOE’s Argonne National Laboratory. Its primary goal of this open-source environment has been to directly support the development of models of social systems. It is agent based and object oriented. A key missing aspect for ecologists is a built-in architecture supporting landscape dynamics. Features of Repast reported on the primary webpage, [http://repast.sourceforge.net/repast_3](http://repast.sourceforge.net/repast_3) are:

- Repast includes a variety of agent templates and examples. However, the toolkit gives users complete flexibility as to how they specify the properties and behaviors of agents.
- Repast is fully object-oriented.
• Repast includes a fully concurrent discrete event scheduler. This scheduler supports both sequential and parallel discrete event operations.
• Repast offers built-in simulation results logging and graphing tools.
• Repast has automated Monte Carlo simulation framework.
• Repast provides a range of two-dimensional agent environments and visualizations.
• Repast allows users to dynamically access and modify agent properties, agent behavioral equations, and model properties at run time.
• Repast includes libraries for genetic algorithms, neural networks, random number generation, and specialized mathematics.
• Repast includes built-in systems dynamics modeling.
• Repast has social network modeling support tools.
• Repast has integrated geographical information systems (GIS) support.
• Repast is fully implemented in a variety of languages including Java and C#.
• Repast models can be developed in many languages including Java, C#, Managed C++, Visual Basic.Net, Managed Lisp, Managed Prolog, and Python scripting.
• Repast is available on virtually all modern computing platforms including Windows, Mac OS, and Linux. The platform support includes both personal computers and large-scale scientific computing clusters.

Repast was selected for a more comprehensive evaluation, which can be found below in the next chapter.

Stella

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Target User</td>
<td>Ecologist</td>
</tr>
<tr>
<td>Lead developer</td>
<td>ISEE</td>
</tr>
<tr>
<td>Component Based?</td>
<td>No</td>
</tr>
<tr>
<td>Source code available?</td>
<td>Yes</td>
</tr>
<tr>
<td>Active development?</td>
<td>Yes</td>
</tr>
<tr>
<td>Agent-based?</td>
<td>No</td>
</tr>
<tr>
<td>GIS oriented?</td>
<td>No</td>
</tr>
<tr>
<td>Multi-purpose?</td>
<td>No</td>
</tr>
<tr>
<td>HPC?</td>
<td>No</td>
</tr>
</tbody>
</table>
Stella is a powerful and easily accessible environment for non-programmers to capture their understandings of the dynamics of systems. It fundamentally supports the notion that systems are defined by amounts of measurable things that increase/decrease based on system interactions over time. It embraces the notion of time in the form of a fixed time-step against which modelers use basic algebra and logic to define how each of the measures change in a time step. There are three interface levels to Stella. The middle level (Figure 1) uses a graphical language to identify the measured things, called stocks or reservoirs, with rectangles; flows with pipes and valves that flow into and out of stocks; and calculations with circles. Arced arrowed lines identify which items influence other items. The lowest level appears upon double-clicking any of the graphical circles and rectangles. Double-clicking a circle reveals an interface for viewing and editing equations in logic and algebra using the variables associated with the incoming arrows. That is, the value associated with circle is a function of the values found at the other end of the incoming arrows. At the highest interface level, the Stella programmer can create an interface for the end-user that provides for run-time adjustments of variables and visualization of run-time changes in selected values (e.g., the bottom of Figure 1).

ERDC researchers have successfully used Stella over the past two decades. It is perfectly suited for rapid model design and development when a system can be viewed as spatially homogeneous and when there are limited interacting objects in the modeled world. For the purpose of the development of an ERDC EMS it is insufficient because of these two limitations. The EMS should support landscape scale models that contain many interacting aspects over a variable landscape.
Figure 1. Sample Stella Interface

Model Builder

<table>
<thead>
<tr>
<th>Feature</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Website</td>
<td>Ecologist</td>
</tr>
<tr>
<td>Target User</td>
<td>ESRI</td>
</tr>
<tr>
<td>Lead developer</td>
<td>No</td>
</tr>
<tr>
<td>Component Based?</td>
<td>No</td>
</tr>
<tr>
<td>Source code available?</td>
<td>Yes</td>
</tr>
<tr>
<td>Active development?</td>
<td>No</td>
</tr>
<tr>
<td>Agent-based?</td>
<td>Yes</td>
</tr>
<tr>
<td>GIS oriented?</td>
<td>No</td>
</tr>
<tr>
<td>Multi-purpose?</td>
<td>No</td>
</tr>
<tr>
<td>HPC?</td>
<td>Windows</td>
</tr>
<tr>
<td>OS</td>
<td>2-D</td>
</tr>
</tbody>
</table>
Model Builder is an ESRI GIS capability that allows for stringing map analysis steps together in graphical form. The graphical nature supports the rapid communication of how the modeling steps are organized and can often be captured in the form of single presentation slides. This type of graphical view of how a model is grossly structured is desired for the ERDC EMS. ESRI has never fully embraced simulation modeling and has instead partnered with third-party developers to provide landscape simulation capabilities. Consequently, the model builder does not directly support dynamic time-step based modeling, nor does it support the notion of GIS vector objects (e.g., points representing individuals, lines representing roads or streams, or areas representing building or habitats) behaving. Therefore, Model Builder does not provide a starting point for the development of the ERDC EMS.

HexSIM

<table>
<thead>
<tr>
<th>Website</th>
<th><a href="http://www.hexsim.net">http://www.hexsim.net</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Target User</td>
<td>Ecologist</td>
</tr>
<tr>
<td>Lead developer</td>
<td>Washington State University</td>
</tr>
<tr>
<td>Component Based?</td>
<td>No</td>
</tr>
<tr>
<td>Source code available?</td>
<td>No</td>
</tr>
<tr>
<td>Active development?</td>
<td>Yes</td>
</tr>
<tr>
<td>Agent-based?</td>
<td>Yes</td>
</tr>
<tr>
<td>GIS oriented?</td>
<td>Yes</td>
</tr>
<tr>
<td>Multi-purpose?</td>
<td>No</td>
</tr>
<tr>
<td>HPC?</td>
<td>No</td>
</tr>
<tr>
<td>OS</td>
<td>Windows</td>
</tr>
<tr>
<td>Dimensions</td>
<td>2-D</td>
</tr>
<tr>
<td>Hierarchy</td>
<td>Landscape</td>
</tr>
<tr>
<td>External connections</td>
<td>GIS</td>
</tr>
</tbody>
</table>
Hexsim is a recent incarnation of the earlier Patch model, also developed by Joshua J. Lawler, University of Washington. It provides an agent-based simulation-modeling environment running on Windows PCs in which all of the traditional software development is provided within the environment. The goal is to simplify modeling to the point where, for a new species and location, the model simply needs to be parameterized and provided with location-specific GIS data. That is the good and bad news is that the modeler does not (or can not) build or modify the model code. Instead, the modeler controls the execution of events:

- Accumulate – Combine information about parts of the system to report as a whole about the system.
- Adjust range parameters – Allows for setting range parameters during a simulation.
- Census – Combines population size data for output.
- Floater creation – Facilitates the separation of members from groups of individuals based on various traits of individuals.
- Generated HexMap
- Interaction – establishes pair-wise interactions among individuals or populations. Supports such things as predator-prey interactions and mating.
- Introduction – Used to add new members to a population.
- Movement (Global, Dispersal, and Exploration) – Used to allow individuals and populations to move throughout the simulation space in varying ways.
- Mutation – Part of the genetics aspect of the model. Used to change the presence of alleles at loci.
- Range Dynamics – Allows populations and individuals to impact their range.
- Reproduction – Mediates the creation of new individuals.
- Set Group Affinity – Facilitates the grouping of individuals.
- Survival – Affects the persistence of individuals over time.
- Transition – Effects the change of the state of individuals (e.g., sick to healthy)

As long as the population being modeled operates with the provided events, HexSim allows for the rapid development of detailed simulations.
of agents behaving on a landscape. It can be very useful for ecosystem management practitioners.

HexSim was not considered for more in-depth analysis because it is not an open-ended modeling environment, is not open source, and is restricted to operation on a single operating system.

**ASCAPE**

<table>
<thead>
<tr>
<th>Website</th>
<th><a href="http://ascape.sourceforge.net/">http://ascape.sourceforge.net/</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Target User</td>
<td>Programmer</td>
</tr>
<tr>
<td>Lead developer</td>
<td>NuTech, Metascape</td>
</tr>
<tr>
<td>Component Based?</td>
<td>No</td>
</tr>
<tr>
<td>Source code available?</td>
<td>Yes</td>
</tr>
<tr>
<td>Active development?</td>
<td>No</td>
</tr>
<tr>
<td>Agent-based?</td>
<td>Yes</td>
</tr>
<tr>
<td>GIS oriented?</td>
<td>No</td>
</tr>
<tr>
<td>Multi-purpose?</td>
<td>No</td>
</tr>
<tr>
<td>HPC?</td>
<td>No</td>
</tr>
<tr>
<td>OS</td>
<td>Java</td>
</tr>
<tr>
<td>Dimensions</td>
<td>1-D</td>
</tr>
<tr>
<td>Hierarchy</td>
<td>Landscape</td>
</tr>
<tr>
<td>External connections</td>
<td></td>
</tr>
<tr>
<td>Documentation</td>
<td>Good</td>
</tr>
<tr>
<td>Language</td>
<td>Java</td>
</tr>
</tbody>
</table>

Ascape is an open-source project that has been developed under the guidance of Miles Parker. It shares many similarities with Repast in that both are Java-based, claim to be descendants of Swarm, use the Eclipse environment for development, support Java programmers, and were developed to support social modeling. Ascape does not support import of GIS data and does not support raster-based simulation modeling and is therefore less valuable for ecological modeling. Like Eclipse, Ascape is developed for Java programmers. It was not considered for further review.
Eclipse

<table>
<thead>
<tr>
<th>Website</th>
<th><a href="http://www.eclipse.org/modeling/">http://www.eclipse.org/modeling/</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Target User</td>
<td>Programmer</td>
</tr>
<tr>
<td>Lead developer</td>
<td>Eclipse Foundation</td>
</tr>
<tr>
<td>Component Based?</td>
<td>No</td>
</tr>
<tr>
<td>Source code available?</td>
<td>Yes</td>
</tr>
<tr>
<td>Active development?</td>
<td>Yes</td>
</tr>
<tr>
<td>Agent-based?</td>
<td>Yes</td>
</tr>
<tr>
<td>GIS oriented?</td>
<td>Yes</td>
</tr>
<tr>
<td>Multi-purpose?</td>
<td>Yes</td>
</tr>
<tr>
<td>HPC?</td>
<td>No</td>
</tr>
<tr>
<td>OS</td>
<td>Java</td>
</tr>
<tr>
<td>Dimensions</td>
<td>1-D</td>
</tr>
<tr>
<td>Hierarchy</td>
<td>Ecosystem; Landscape; Population; Individual; Chemical</td>
</tr>
<tr>
<td>External connections</td>
<td></td>
</tr>
<tr>
<td>Documentation</td>
<td>Excellent</td>
</tr>
<tr>
<td>Language</td>
<td>Java</td>
</tr>
</tbody>
</table>

Eclipse is a powerful and popular open-source Integrated Development Environment (IDE), developed primarily to support Java programming. It is fundamental to the Repast environment and can be used in the development of OMS components. It is also a general purpose Java design and development environment and therefore, a powerful environment for building ecological models. It was not selected for further consideration as a stand-alone EMS environment, but is considered through the deeper Repast evaluation.

Others

Several other very good model development environments were not evaluated due to their proprietary nature. GoldSim (http://www.goldsim.com) is similar in nature to Stella as a one-dimensional system dynamics modeling system. AnyLogic (http://www.xjtek.com/) is very similar to Netlogo, but proprietary and expensive. The RAMAS (http://www.ramas.com/ramas.htm) metapopulation modeling capabilities have been used in metapopulation-based military installation population viabilities analyses. It looks at
groups of interacting populations and considers changes in threats to each population and genetic exchange among the populations to predict the potential for complete loss of the meta-population over time.
Appendix B: Initial Model Integration Evaluations

XMS

In support of hydrologic modeling, ERDC-CHL integrated sets of hydrologic codes to create the Watershed Modeling System (WMS), the Surface-water Modeling System (SMS) and the Groundwater Modeling System (GMS). Collectively these are referred to as the XMS systems. Each suite of models were placed behind a common user interface and were modified so that they would read/write data files that allowed a hydrologic modeler to focus on the modeling needs without worrying about modifying the output from one analysis into the inputs of a next analysis. With the XMS systems, modelers could download land cover, bathymetry, and digital elevation maps; process those maps with GIS commands; set boundary conditions; and send the results to the various models. This provided a major step forward in the integration of hydrologic codes, but did not solve the data exchange in a generic way. Indeed, each of the suites supported data exchange formats that cannot be shared among the suites. More recent efforts have sought to solve the challenge of linking watershed, surface water, and groundwater models.

XMDF/XDMF

The Extensible Model Definition Format (XMDF) was developed by ERDC and Brigham Young University (Fife 2006). The Extensible Data Model and Format (XDMF) was developed at the Max Plank institute. Both provide solutions to store, access, and share data representing the state of a system among a suite of models that act on different parts of that system. XDMF was designed to support suites of computer models running on HPC systems. Both approaches chose to use the HDF5 (Hierarchical Data Format) system for storing large data sets. HDF can be thought of as support for an entire file system within a file. The primary advantages are that an HDF file can be shared across all operating systems, and many different files can be managed as a single file. A model then uses the HDF5 software libraries for reading/writing data from these files, and the specific contents of that file are under the complete control of the modeler. An HDF5 file might contain a structure to store all of the inputs associated
with a modeling exercise, results generated during modeling runs, and results from multiple runs.

Both XMDF and XDMF are systems to store and organize data. Both could be utilized by any modeling system.

**Conceptual Model Builder**

The Conceptual Model Builder (CMB) is a set of tools developed by ERDC to provide modelers with an environment in which they can build and initialize scenes for multiple, discipline-specific numerical models. CMB was originally developed to create large computational scenes for groundwater modeling but has gained traction in use in several disciplines. The CMB makes it possible to create a scene from LIDAR, TINs, Object files, etc. The user can tag identify key features, tag boundary conditions, and tag material regions. The CMB is customizable for specific models --- a template can be created and loaded which determines what input parameters the user sees and controls as well as what output files are created by CMB. This flexibility makes CMB attractive to use as a front-end for a component based modeling system. In one system, the user will be able to create the computational domain and mesh it (if necessary), specify the initial values for model parameters, and create the input files for the EMS modeling system.

CMB is platform independent and is built on a kernel of ParaView, a widely used data analysis and visualization tool.

**CCA/CSDMS**

The Community Surface Dynamics Modeling System (CSDMS) is a NSF-sponsored community of earth surface process modelers. In 2010, CSDMS hosted 164 earth surface process models (terrestrial, coastal, marine, hydrology, carbonate, and climate) in its repository, provides a model linking environment and approach for the high performance computing (HPC) community. The CSDMS Modeling Tool, running on PC, Mac, and Linux) provides an environment to assemble and run model components on the CSDMS HPC. These models are linkable through the CSDMS adoption of a high-performance version of the common component architecture (Bernholdt 2004). The development of a high performance CCA has involved collaboration among researchers at several DOE labs (Argonne, Oak Ridge, Pacific Northwest, Sandia, and Lawrence Livermore),
universities, and commercial entities. The Common Component Architecture (CCA) forum began in 1998 to create a component model to meet the needs of high-performance scientific computing.

The elements of the CCA model are components, units of functionality in software that can be combined together; ports, “abstract interfaces through which components interact”; and frameworks, through which CCA components are assembled into and executed as models. CCA uses Babel, an interface definition language, which supports component interaction via ports. It relies on the Scientific Interface Definition Language (SIDL) for defining interfaces through defined types and declared methods. It uses Ccaffeine to orchestrate a set of SCMD (Single Component Multiple Data) components on a parallel machine, relying on MPI or PVM to spawn processes as needed.

Like OpenMI and FRAMES, legacy models should be wrappered in a specific manner to allow for the resulting component to operate alongside other components. Unlike OpenMI and FRAMES, CCA/CSDMS supports simultaneous operation of models that interact with one-another during simulations.

OpenMI

The Open Modelling Interface (OpenMI) is the result of a European Union funded effort to support the linking of models developed across Europe that will assist with complying with the EU Water Framework Directive in 2000. OpenMI has been a joint effort among European commercial and state research institutes that is allowing disparate water models to be connected through their inputs and outputs. The primary goal of OpenMI is to capture existing stand-alone models (particularly water models) as components of the larger system that can be linked. It is not intended to support the design and development of new models.

A fundamental design philosophy behind OpenMI is that models, in response to requests from other models, generate information. In other approaches, the modeling process begins with the most upstream model running and dumping its results in a format that can be read by the next model, and so on. In OpenMI, the process begins with the identification of desired final results. This results in the most downstream model being initiated first. But, to proceed, this model should have certain input information, which may come from another model. That model receives a
request for the information and to comply it may request information from another model and so on. Models that have been developed to respond to requests in this manner are OpenMI components.

OpenMI establishes the form and format of the information exchange as exchange items, which are composed of quantities and element sets. The quantity is a class that consists of an identification value, a description, the value type, the units, and the dimension. Quantities in OpenMI include length, mass, time, current, temperature, amounts, luminosity, and currency. Element sets contain a set of elements, ordered to facilitate retrieval. Elements have a position in space and are associated with the value of the quantity at that location. Element sets can be two- and three-dimensional points, lines, polylines, and polygons. Time is handled as timestamps associated with an element set that indicate a point or range in time.

Existing models should be wrapped within an OpenMI compliant wrapper that facilitates the communication among models using the OpenMI defined element sets and information requests and the internal workings of the model. The wrappers should communicate with the model for initialization, inspection and configuration, preparation for execution, execution, completion, and post-execution cleanup. Execution is accomplished as steps through time, which allows models to run simultaneously and potentially over a network.

OpenMI is a powerful and well-funded modern effort. An expanded review of OpenMI is included in a later section.

FRAMES

The Framework for Risk Analysis in Multimedia Environmental Systems (FRAMES) is a product of DOE’s Pacific Northwest National Laboratory (PNNL). It has been used by a number of agencies, including the EPA and ERDC, for model linking and addresses the same challenge as OpenMI – linking existing models.

Like OpenMI, the FRAMES philosophy is that there is great value in adopting existing established models and giving them new life as linkable components. Perhaps the greatest fundamental difference is that FRAMES compliant models are run to completion before results are passed to the next downstream model, while OpenMI compliant models may run simultaneously by passing results for each time step to the downstream
models. Neither system facilitates the supporting of feedback loops that involve passing output from downstream models to upstream models.

To make an existing model FRAMES compliant, the model should be wrapped in code that can read/write FRAMES-defined data, communicate that information into/out-of the model, and appropriately initialize and run the model. Data exchange is handled through sets of Primary Data Communication Files (PCDF). Every set includes an error file, a global data input file, and a description file. Then, there are data PCDFs that are associated with different environmental assessments, such as air flux, atmospheric transport, ecological effects, water flux, water concentration, and others.

Like OpenMI, FRAMES supports the precise definition of units used in the data files to ensure that the model-to-model connections are appropriate.

Once models are appropriately encapsulated within FRAMES-compliant software and can read/write FRAMES data files, they become FRAMES modules. The FRAMES system provides a graphical interface that gives a model developer access to detailed information about modules and the ability to assemble, parameterize, and run sets of modules as a single model.

ERDC has significant experience using FRAMES as the integration environment behind the Army Risk Assessment Modeling System (ARAMS) (Lloyd 2007).

**OMS-3**

The Object Modeling System, version 3 (OMS-3) is a very active US Department of Agriculture (DOA) solution to the challenge of integrating models. Version 3 is a complete rewrite of OMS within the Java language. It supports the encapsulation of legacy codes for the creation of coarse-grained components, but also embraces the development of very fine-grained (i.e. single-purpose) components. The motivation behind the design and development of OMS is to facilitate the linking of models that represent the dynamics that drive agricultural systems.

OMS-3 is has two main aspects. First, and for the interest of ERDC, it is a model component design, development, linking, and execution environment. Second, it is offers several collections of developed components
created to meet the needs of specific users. DOA OMS-3 components include:

- Hydrology and rainfall-runoff
- Erosion, sediment transport, phosphorus, and nitrogen
- Range-Livestock
- Distributed watershed models
- Isotope model

All ongoing USDA / CSU model developments is open source and available under LGPL. Nearly 100 components are currently publicly available on the OMS-3 website. Additionally, an Italian research group has adapted the ERDC GRASS GIS capabilities into an OMS-3 library called JGrass.

The aspect of OMS-3 most interesting to the ERDC EMS is the provision of a general-purpose modeling framework for developing model components and integrating them into models. Key aspects of OMS-3 as a component development and integration environment include:

- Platform independence
- Cloud/cluster computing support (Linux)
- Components can be written in Java, C, C++, Fortran, NetLogo
- Components can be fine grained physical processes or whole and complex models
- Components are run as separate threads
- Requires low invasiveness to existing models
- Can encapsulate existing ERDC models
- Can support full dynamic feedback among model components
- Allows the quick addition of ad-hoc model components
- Can reveal meta-data about components to help modelers best select and connect components
- Supports high-performance computing (room for improvement: in development)
- Fully promoting component based model development
- Binary reuse, source level component in various ‘scientific’ languages (Java, Fortran, C, NetLogo, R(2011))
- Currently about 250+ modeling components
- Fully metadata-based noninvasive component development
- Facilitates multi language component integration
• Simple simulation development (basic, calibration, sensitivity analysis, and ensemble runs) using a simulation DSL.
• Auto-documentation, auto-testing, auto-scale of components, and models
• Integrates into various IDEs for development (e.g., Netbeans, Eclipse, and IntelliJ)
• Support for importing data and maps

The initial review of OMS-3 led to a more in-depth analysis, which is provided in a later section of this document.

Summary

This section covered the cursory evaluation of a

<table>
<thead>
<tr>
<th>Name</th>
<th>XMS</th>
<th>XMDF</th>
<th>CMB</th>
<th>CSDMS</th>
<th>OpenMI</th>
<th>FRAMES</th>
<th>XMS</th>
<th>OMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer</td>
<td>ERDC/BYU</td>
<td>ERDC/BYU</td>
<td>ERDC</td>
<td>CSU</td>
<td>European Union</td>
<td>Dept of Energy</td>
<td>ERDC</td>
<td>Dept of Agriculture</td>
</tr>
<tr>
<td>Active?</td>
<td>Yes</td>
<td>Yes</td>
<td>YES</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>HPC?</td>
<td>No</td>
<td>Yes</td>
<td>Only</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Dynamic?</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Open?</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Appendix C: Repast, Repast Simphony, and ReLogo

Authors: David Stigberg and Todd Swannack

Introduction

Repast Beginnings

The agent-based modeling toolkit and framework, Repast (the Recursive Porous Agent Simulation Toolkit), is one of the pioneering spatially-explicit agent-based modeling tools. It originated at the University of Chicago, first appearing in 2000. Since shortly after its inception, Repast has been maintained and continually developed at Argonne National Laboratory; since 2004, it has been managed by the non-profit volunteer Repast Organization for Architecture and Development (ROAD).¹

Repast began as a free and open-source Java-based implementation, and Java remains a principal focus to the present. But it also has taken some different directions as it has developed. By 2004, Repast 3.0 was available in three different versions: along with a Java-based version, now renamed Repast J; Repast.Net, an implementation in C# for Microsoft .Net modelers; and Repast Py, which enabled modelers with limited programming experience to use the high-level Python programming language to develop more basic models.²

Repast Simphony

In 2007, a new incarnation of Repast appeared, renamed Repast Simphony (also called Repast S). Repast Simphony saw a dramatic growth of the Repast code base and the Repast Java API, in part through the integration of several third-party code libraries, and while many core elements of Repast 3 were retained, the code itself was largely reorganized.

¹ At the outset, I wish to express my thanks to the generous help provided to me by Repasts developers at the Center for Complex Adaptive Agent Systems Simulation at the Argonne National Laborarory. In particular, I wish to thank Eric Tatara, and Jonathan Ozik, the lead developer of Repast ReLogo at Argonne, for their patient responses to my many questions.

² See North et al. (2006), which discusses all three versions of Repast 3.0.. Railsback et al. (2006) is an important and rigorous study, comparing pre-Simphony Repast (Repast 3.1) with NetLogo, and with two other agent-based modeling platforms: MASON, and Swarm (Objective-C and Java Swarm versions).
and revised. There were also several substantive innovations in Repast Simphony, many persisting to the present. The most important of these innovations are listed below; some will be treated later:

- Integration with the widely used Eclipse integrated development environment (IDE), with the creation of a powerful Repast Simphony “plug-in” for Eclipse. A set of “visual” development tools that are designed to help modelers develop their models with less need for traditional code writing.
- Ability to easily connect with various third party applications (MATLAB, GRASS, MS Excel and other spreadsheets, etc.)
- Addition of the Groovy programming language to the Repast toolkit. (Recently, as we shall see, the role of Groovy has become much more important, with the introduction of Repast ReLogo.)
- New model framework components and resources, with several key elements and concepts, including, among others: contexts, projections, queries, Java annotations for watchers and scheduling, and scenarios.

ReLogo

Java seems to be the language of choice of a majority of extant agent-based modeling systems, but there are alternatives to such systems¹. Perhaps the most important and widely used of these is NetLogo. NetLogo is written in Java and Scala, but users write their models with the powerful but easy-to-use NetLogo language. NetLogo, itself a principal candidate for use as part of an EMS, is given separate coverage elsewhere in this document.

Recently, Repast developers at Argonne have added a new capability to Repast Simphony called ReLogo. ReLogo aims to capture the ease of use and rapid development capability of a Logo-based language (like Netlogo), while at the same time making all the other powerful features of Repast Simphony available to the ReLogo modeler. ReLogo first appeared with the release of Repast Simphony 2.0, beta, in December 2010. Due to the promise of its particular combination of strengths, and, in the present context, a potentially attractive alternative to NetLogo, ReLogo will form the focus of our discussion of the Repast platform to follow².

¹ For some recent surveys, see Nikolai and Madey (2009), and the many useful links at http://www.swarm.org/index.php/Software_Reviews.
² Also released at the end of 2010 was Repast for High Performance Computing 1.0.1 beta (RepastHPC). Repast HPC is based on C++ and provides, along with the performance speed of C++, an API that allows modelers fine-grained control over the distributed processing of their models. Repast HPC should be of interest to advanced modelers with substantial prior Repast experience.
ReLogo’s attractiveness for the ecological modeler

ReLogo and NetLogo

One of the most significant aspects of ReLogo as a language is its similarity to NetLogo. ReLogo code in fact looks very much like NetLogo code. Most of the various NetLogo commands have similar counterparts in ReLogo, albeit with differences in detail and syntax that derive from Groovy, the language upon which NetLogo is based (see below), and Java.

Like NetLogo, the ReLogo language is economical and elegant, expressive, and is intended to be relatively simple and easy to use. It lends itself to rapid development, and should find much of the same appeal for relatively inexperienced modelers as NetLogo. At the same time, and also like NetLogo, it is capable of model development of sophistication and complexity. The following figure shows snippets of code, in ReLogo, NetLogo, and Java, that rotate and then move an agent a specified distance (1.5 units) towards a target patch on a grid. It provides a striking illustration of the similarity between ReLogo and NetLogo, and at the same time, the simplicity and expressivity of both these examples compared to the much more complicated Java code that accomplishes the same thing:\footnote{The Java and ReLogo examoles are adaptations from the Repast Java and ReLogo “Getting Started” tutorials. Note that the Java example has a small extra complicaton, in that it checks to make sure the target space and the agent’s starting location aren’t the same. It is otherwise complicated by the need for explicit conversion between the floating-point values of the mobile agent’s two-dimensional continuous space, and the integer values of the gridded space.}:

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure.png}
\caption{Comparison of ReLogo, NetLogo, and Java code for rotating and moving an agent.}
\end{figure}
The NetLogo import utility

A very important part of the Repast ReLogo package is a tool for the import and conversion of NetLogo models to working ReLogo models. This tool provides, for the NetLogo modeler, a superb entrance into the world of ReLogo and Repast. And for modelers generally, whether or not they have prior experience with NetLogo, it provides the promise of access to the huge library of NetLogo sample models that can be used to jumpstart the model building process in ReLogo, providing working models for study, and prototypes that can be modified and expanded to create new models in the Repast ReLogo environment.

ReLogo and Groovy

ReLogo is by no means a carbon copy of NetLogo or any of the other members of the Logo family of languages. ReLogo is a fully object-
oriented, domain specific language, written in the Groovy programming language (Ozik, Relogo getting started guide). From the developer’s perspective, ReLogo is a kind of custom extension to Groovy. ReLogo code is written in .groovy source code files. ReLogo commands and procedures that are available to modelers are found in Groovy APIs (Application Programmer Interfaces) alongside other Groovy APIs. The important point here is that ReLogo is Groovy, and as such can do all the things that Groovy can do. It allows the ReLogo modeler to seamlessly combine ReLogo and other Groovy classes and methods, and these in turn can be integrated seamlessly with Java code. We will return to this point later.1

Java, Repast, and ReLogo

The Java programming language has undergone steady development since its origins in the early 1990s. Today, the core Java language (currently Java 6 is most well-established) embraces a huge repertory of functions in many integrated code libraries of classes whose methods are freely available to all through well documented APIs. In addition to the core language, third-party libraries, a great many of which are also freely available, provide additional specialized functionality through their own APIs.

Repast Simphony is written in Java, with the creation of much new Java code, but it also integrates many third-party Java libraries. These resources provide sophisticated functionality for developers of Repast models, in such areas as 2D and 3D graphics, user interface development, network analysis, randomization and statistics, GIS capabilities, and more. Since ReLogo (and Groovy), can be seamlessly integrated with Java, the promise of Repast is that all these Java-based resources are available to the skilled ReLogo and Groovy modeler, just as they are to traditional Repast Java programmers.2

---

1 It is beyond the scope of the current treatment, but there are several reasons why Groovy was chosen as the language upon which to build ReLogo. Groovy has a number of distinctive strengths, some of these are realized in ReLogo proper, but there is also much more that can be done with Groovy—in Repast and while writing ReLogo models—than is present in ReLogo per se. The interested reader will want to see Ozik et al. (2007) for a discussion of Groovy and its use in creating ReLogo; also the official Groovy website, http://groovy.codehaus.org; and, among the several books on Groovy, especially Koenig et al, (2007) (second edition scheduled for Summer 2012).

2 Since Repast is open-source, Repast code libraries can be modified by sophisticated and enterprising users to satisfy the particular needs of the their models, to help integrate their Repast models with legacy software, or even to contribute to the ongoing development of the Repast Simphony codebase.
ReLogo and the object-oriented paradigm

It has been noted that ReLogo, like Groovy and Java is a fully-object oriented language. Encapsulation, inheritance, the use of class interfaces, and other object-oriented techniques are available to the ReLogo modeler and offer the same strengths as they do for modelers in languages like Groovy and Java.

Object-oriented programming seems especially suitable for agent-based modeling applications, where, e.g., the encapsulation of an object’s properties and methods in classes provide a quite natural and intuitive way to capture real-world model agents’ characteristics and behavior in code, and where the use of inheritance can provide a powerful and economical way to capture both the similarities and differences among agents in complex multi-agent modeling applications. And, for agent-based modeling as for other kinds of programming applications, these and other object-oriented techniques can go far to encourage program modularity, clarity, maintainability, and reusability.

The object-oriented nature of ReLogo, which significantly differentiates it from NetLogo, will be seen as an asset for those programmers with the skills and experience to take advantage of it. But it also represents potential difficulties for the novice programmer, difficulties that are foreign to the NetLogo experience.

Eclipse and the Repast Simphony “plug-in”

In the standard installation, Repast Simphony (including ReLogo) is made available as a plug-in, a software package written to facilitate Repast model development, which sits on top of Eclipse, the very popular Java-based IDE (integrated development environment). Repast, the Repast plug-in, and Eclipse are all free.¹ Eclipse and the Repast plug-in together provide a complete environment for developing, running, testing, and maintaining Repast models written in Java, Groovy, and/or ReLogo. While Repast models could be written outside of Eclipse, it would be much more difficult, and perhaps little would be gained in doing so.

¹ For installation, see the links under “Repast Simphony” at http://repast.sourceforge.net/download.html.
Working with Repast in Eclipse involves two distinct phases, first, code-writing in the Eclipse IDE proper, followed by, second, the launching of the Repast runtime environment, where a model is executed (and where certain aspects of the model are assembled and modified). The two environments roughly parallel NetLogo’s Procedures and Interface work areas, respectively.

Figures 2 and 3 show a simple ReLogo model (“Zombies”) as it appears in, first, the Eclipse IDE, and second, in the Repast Runtime environment after being launched from the IDE.¹

---

¹ Figure 2 shows the two main panels of the Eclipse IDE, a Package Explorer on the left, and a source editing screen on the right. The Package Explorer shows the roots of two versions of a “Feralpigs” model, and an expanded Package tree for the tutorial-derived Zombies model. The nine .groovy source files in the zombies.relogo package are listed, along with a listing of various shapes available for graphic display. The right panel shows the Zombie UserObserver.groovy source file open for editing. Finally, note that the Package explorer does not display all the Zombies files here. A user-setTable filter has screened out all but the most important ones.

Figure 3 shows the Repast Runtime screen, after launch of the Zombies model from the Eclipse IDE, and displaying the Zombies model in the middle of a run.
With Repast, thanks to the Eclipse plug-in, many tasks, especially with simple models, are handled easily, just as they are with NetLogo. But there is always at least a little and often a lot more for the Repast and ReLogo modeler to do, in terms of setting up and configuring graphic input and output display components, data loaders and “outputters” of various kinds, editing context and scenario .xml files, and more. But for the skilled and imaginative developer, these tasks can be seen as opportunities for fruitful customization of a model’s appearance and functionality, rather than just extra work.

**The Repast Framework: Contexts and Projections**

There are many features of the Repast framework and programming toolkit that help in the creation of complex and ambitious models, whether the modeler is working in Java, Groovy, or ReLogo. **Contexts** and *Projections* are two of these.

Repast provides substantial support—via context and context-related classes, and a context.xml file—for the use of multiple contexts in a single model. A *context* in Repast is a container, an object that contains other
objects—mobile and stationary agents, link agents, data layers, etc. Contexts can be nested, so contexts can contain sub-contexts. Contexts can be programmed to have attributes and behavior, and sub-contexts can inherit the features of the contexts above them while adding their own attributes and behavior. Agents of course can interact with other agents, but they also can interact with different contexts as they move among them.

*Projections* in Repast serve to define and maintain relationships among agents and other objects within a context. There are grid, continuous space, and network projections, and a geography projection for GIS applications. Projections have different capabilities, appropriate to the projection type, for working with agents—placing them, retrieving them, and linking them to one another. Agents may be associated with multiple projections in a particular context. If the complexity provided by the defined projections in the Repast API is not sufficient, *custom* projections can always be created by the ambitious developer.¹

This brief discussion of Repast contexts and projections serves to reinforce a general point made throughout this section: Repast makes a rich set of programming resources, and in Eclipse, a powerful development environment, available to the ReLogo modeler. The complexity and diversity of these resources, along with several of the more complex aspects of the ReLogo language itself, inevitably steepen and lengthen the learning curve for new ReLogo modelers. But just as many useful and interesting ReLogo and Repast models can be created with a single context and a single projection, more intricate use of Repast’s tools can be learned gradually, or even ignored altogether. The resources are there when the modeler is ready for them.

**ReLogo testing and evaluation**

Relogo and Repast were evaluated by T. Swannack and D. Stigberg. Each reviewer spent time independently examining ReLogo, engaging in model-creating activities and working with the Netlogo-ReLogo import utility. The authors bring different experience to bear on this effort. Swannack is an active research ecologist and ecological modeler with relatively modest programming experience. Swannack’s main goal was to evaluate how

¹ For further details about Repast contexts and projections, see Howe et al., 2006. A relatively simple but very instructive model, both for its use of Repast’s vector GIS capabilities and for its clear use of multiple contexts, is RepastCity, written in Java by Nick Malleson (2011). It is available for download from http://portal.ncess.ac.uk/wiki/site/mass/repastcity.html
easily new modelers, with little or no Logo programming experience, could
develop models using Repast S, including ReLogo. Stigberg is primarily a
programmer and software developer with several years experience with
Java and other languages, and has modest NetLogo programming
experience. Stigberg has roughly a year of experience working with Java
within Repast, limited Groovy experience, and hands-on experience with
ReLogo only beginning with the release of its release in December 2010.
Principal programming and testing activities were as follows:

- Both authors carefully went through the ReLogo Getting Started
tutorial, using it to create working versions of the sample Zombies
ReLogo model. Swannack attempted to expand the Zombies model
with new ReLogo code.
- Stigberg tested the NetLogo-ReLogo import utility with several small
NetLogo sample models, and also with two complex and large-scale
ecological models developed under ERDC auspices: a Feral Hogs
Population Control model (Burton, 2012) and a Fort Stewart Gopher
Tortoise model (Tuberville 2012).
- Stigberg attempted a partial adaptation of the open-source Java library
that forms the core of the NetLogo GIS Extension, which provides most
of the GIS capability possessed by NetLogo. This was motivated
principally since GIS capabilities, especially raster GIS, are not
abundant in Repast or ReLogo, and one of the imported NetLogo
models (the Gopher Tortoise model) makes heavy use of these
capabilities via the GIS Extension. Apart from simply being able to get
the Gopher Tortoise model to work in ReLogo, there were two
additional goals:
  - to try to reach an assessment of the effort required to add
    comprehensive GIS capability to ReLogo
  - to test the ability to combine Java and ReLogo code in ReLogo
    models.

Writing ReLogo models and importing NetLogo models

Experience with the “Getting Started” Tutorial and the “Zombies” Model

The Getting Started tutorial provides a gateway into the ReLogo language.
The example in the tutorial consists of developing a spatially-explicit
disease model in ReLogo and is very applicable for ecological modeling.
The example contains two populations: humans and zombies, with the
latter being the disease vector that can spread to the former. Each population is treated as a class and the tutorial/guide provides step-by-step instructions on how to create and implement the model, including all of the necessary code. The model is relatively easy to implement, however, the guide does not provide any troubleshooting guidance in case users make mistakes (e.g., as TS did when first programming the model). Error messages were not intuitive, nor did they necessarily guide users to problematic lines of code. The zombie model was going to be used as a template for other models. The lack of detailed, user-friendly documentation describing the ReLogo language really hampered other models being developed using just ReLogo. Further, there are very few examples coded in ReLogo to use as learning tools, however, that should change after summer 2011 because Argonne was awarded several Google Summer of Code interns, who were going to program more example models. We found that importing Netlogo examples into Repast was the easiest way to work with models within ReLogo, but there were some issues with the import module (see below).

Importing NetLogo Models

While ReLogo is any many ways simpler, even vastly simpler than traditional Java, it is not as simple as NetLogo. With the Zombies model in the Getting Started tutorial, one gets merely a taste of the complexities of ReLogo, relative to NetLogo—in terms of syntax and notational details, the use of multiple, rather than a single source file, and only the mildest suggestion of the subtleties and potential of object-orientation. All of these are factors sure to come into play with the creation of more substantial and more sophisticated models.

Rather than attempt to write such models in ReLogo from scratch, we availed ourselves of ReLogo’s NetLogo import utility—an essential tool for learning about and testing ReLogo—to attempt to bring some more complex models from NetLogo into ReLogo and the Eclipse environment. A substantial amount of time, therefore, was spent with the importer, which does in fact work, and to a large extent works very well. But we also discovered some wrinkles and bugs in the import process:

- NetLogo is a more forgiving language than ReLogo (or Groovy, or Java), and many constructs that are legal in NetLogo are not in ReLogo. Thus, for example, variable names like #_birds, %_rainfall, or even numBaits/100 are perfectly acceptable in NetLogo, but they
are not in ReLogo, and should be changed before attempting an import (e.g., numBirds, pct_rainfall, baitsDividedby100.) 

- Similarly, complex NetLogo statements can often be written with only minimal bracketing (parentheses, curly-, or square-braces), but in several cases, such imports raised syntax error warnings in ReLogo, and had to be analyzed and rewritten.
- Among other relatively small bugs, a consistent mistranslation of expressions involving the while keyword was an irritation but easily corrected once discovered.
- NetLogo comments are not carried over in the import process. While understandable, this represents a potentially serious loss of code documentation, the remedy for which—manual comment insertion—could be a significant nuisance.
- The importer does not import NetLogo extension references and commands; such lines are simply omitted. However, as a by-product, in some cases the resulting imported object proved to be seriously corrupted. Until this is remedied, any extension references should be commented out before attempting import.
- Finally, one significant bug in ReLogo itself was discovered during this exercise, and that is the handling of the NetLogo myself primitive, which in some cases does not give expected results.

It should be noted that none of the bug and problems revealed in our testing are fatal; all could be worked around relatively easily. Also, none of the above are surprising given the beta nature of this complex product. It is expected that these problems will be corrected in subsequent releases. But, as a practical matter, the time spent discovering and attempting to fix such problems did prevent us from getting as far in our testing as we would have liked, and it is certainly likely that there are problems we missed.

However, importation from NetLogo has its limits regarding what it can reveal of ReLogo’s potential. There are matters of syntax in NetLogo, for example, which, given ReLogo’s grounding in Groovy (and Java), would idiomatically be handled differently in ReLogo. And since some of the advanced capabilities available to the ReLogo modeler, such as the use of multiple contexts, and multi-levels of inheritance among agents and other objects, have no counterpart in NetLogo, the import tool will not provide

---

1 Variable names with opening capital letter, e.g., “MasterlessPigs” imported into ReLogo produce a groovy.lang.MissingPropertyException at runtime in ReLogo. This does not seem to be a limitation of the Groovy language, and may be a ReLogo bug.
instruction in these areas. For this, one should hope for more, and more complex, sample models from the Repast and ReLogo developers in a subsequent release.

NetLogo extensions and adapting the NetLogo GIS extension to ReLogo

As noted earlier, one of our import exercises involved a complex ecological model (Fort Stewart Gopher Tortoise) from NetLogo that uses the NetLogo GIS extension. The import utility does not support NetLogo extensions, and so we used this model as a vehicle for testing an adaptation of the Java source API used in the NetLogo GIS extension to ReLogo. Suffice it to say that, acknowledging that our adaptation was quite limited in scope—it did not involve vector GIS components at all, nor did it involve several of the raster-specific GIS procedures—the experience, as far as it went, was successful. If nothing else, the exercise demonstrated the general feasibility of adding, with relatively modest effort, new Java libraries (jars) to ReLogo applications within Eclipse. It also provided a good demonstration of the easy combinability of Java, Groovy, and ReLogo code in the same .groovy source file. Third, with regard to GIS in particular, it suggests that, given the availability of open-source models for GIS procedures, it should be relatively easy to supply some of the basic, essential GIS capabilities that currently ReLogo lacks. But finally, the makeshift nature of our little test should be emphasized. Much more valuable would be an adaption of this or other comprehensive GIS capability to ReLogo, an adaptation that would also have the kind of easy-to-use use, NetLogo-like commands that the NetLogo GIS extension uses and which are otherwise characteristic of ReLogo. One hopes that such a development will materialize at Argonne in the future.

Repast and ReLogo: Summary of Capabilities and Strengths

The following section gives brief assessments of the capabilities and strengths of repast Simphony, and in particular Repast ReLogo, in each of several categories. Together they are intended to provide a fairly comprehensive basis for evaluation of the software. Each section received a letter grade on the A-F scale.

---

3 The level of our accomplishment in this adaptation should not be overstated. Besides being limited in scope, it did not involve the creation of a simple set of reusable commands for GIS operations like those for other ReLogo operations (or, like those of the native NetLogo GIS extension itself). Such a GIS capability, one that is comprehensive and that includes a true ReLogo command interface would be a wonderful addition to Repast Simphony.
**Strength of System, on-going development**

Development activity: Different versions of Repast have been under development since initial release in 2000. Each different version of Repast has gone through multiple releases. More frequent updates than have been typical would be desirable; in particular an update of the Repast Simphony 2.0 beta from December 2010 is eagerly awaited.

Ownership: Argonne National Laboratory (DOE) has been responsible for maintaining Repast since the beginning. While Repast has always been free and open-source, a team at Argonne has almost exclusively carried out development of Repast itself. However, collaborative development partnerships with the Argonne team are possible.

Availability and extensibility: Repast Simphony (including ReLogo) is freely available and open-source (extending to most if of its included third-party components). It is extensible throughout, via modification of core framework and modeling libraries (where license permits).

User community: judging by the activity of the “repast-interest” user forum, Repast has long enjoyed the interest of a modest but steady stream of active users. It may be significant that, apart from contributions by Argonne developers, new-user questions and comments dominate the user forum.1

**Score: A-**

**Support for spatially-explicit modeling**

Kinds of modeling supported: emphasis is on spatially-explicit, agent-based modeling, although systems dynamics modeling is supported as well. Stationary, mobile, and network agents of various kinds can be defined (in ReLogo, as in NetLogo, such agents are patches, turtles (and specific turtle “breeds”), and links. The general Repast agent-based model-framework is time-step oriented, and allows both fine-grained schedule control and multi-thread step control.

Spatial dimensions supported: both 2-D and 3-D spatial modeling are supported. Grid (integer) and Continuous (n-dimensional floating-point)

---

1 For Repast source code files, information about project activity, Repast user forum archives, and more, see the various links available at the Repast SourceForge project page at http://sourceforge.net/projects/repast/
projections are implemented, and may be combined in the same model. A Geographic projection supports both Vector and Raster GIS implementations (although the Raster GIS implementation is quite limited).

Repast and ReLogo implement complex network modeling, supported by full integration with the JUNG network modeling library.

**Score:** A-

**Library of existing models supporting ecological analysis**

The Repast Simphony framework, with its support for multiple, hierarchical relationships among components does lend itself to complex ecological analysis, but the lack of a substantial native raster GIS capability is a handicap.

Sample models. Presumably since Repast Simphony 2.0 is a beta release, only a quite small number of models, most deriving from earlier Repast Simphony 1 releases, are available for import into release 2.0. None of these models have been converted to ReLogo, yet.

For ReLogo, there is compensation in the fact that perhaps a majority of the literally hundreds of sample NetLogo models, many of them ecologically oriented, can be imported relatively easily into ReLogo. These models do not take advantage of specific Repast ReLogo features that have no direct counterparts in NetLogo.

**Score:** C-

**Opportunities for connecting to other models**

Repast ReLogo and Java-coded models can be embedded in and controlled by other programs, and can be combined, and connected with other models. In all cases, this will involve some user programming.

**Score:** A-
Support for component-based modeling

There is no explicit framework for content-oriented component-based modeling, and use of Repast to create such a framework would involve a very substantial programming effort.

However, for those with adequate Java (or Groovy, and possibly ReLogo) programming strength, Repast Simphony might be an ideal tool for the creation of the components that could be used in such a system.

Score: D (for use as a component-based framework)
B (for component development)

Platform independence

Complete installation packages for Windows and Mac OS X containing Repast, the Repast Eclipse plug-in, Java, etc. are freely available and easy to install. Linux installation is also freely available, but involves some assembly on the user’s part.

Repast HPC (Repast for High Performance Computing) is also available for Windows, Mac OS X, Linux, and other Unix systems.

For documentation and links to the installation packages, see http://repast.sourceforge.net/

Score: A

Connections to external data analysis tools

Automated connections to a number of external tools are implemented, including MATLAB (computational mathematics), VisAD (scientific visualization), *ORA (network analysis, from Carnegie-Mellon), the R statistics package, Excel and other spreadsheets, GRASS (open-source raster and vector GIS), and several others.

In most cases, the connections are rudimentary although useful, e.g. launching an instance of the target application with Repast output data as input. Most of the external tools should be installed separately and any manipulations of the model data occur outside the Eclipse environment.
Score: B-

**Easy entry for non-programmers?**

There are three languages used in the Repast environment, Relogo, Groovy, and Java. Java and Groovy are languages used by professional programmers. There is a Repast Java “Flow Chart” interface (user creates flow-charts which are automatically converted to working Groovy code), but that does require some programming experience. In general, substantial programming experience with modern Java (or Groovy) and with object-oriented techniques is required. Even for experienced programmers there will be a significant learning curve (due to both the inherent complexity of the Repast framework and inadequate documentation).

ReLogo provides a slightly easier entry into the Repast environment, but some programming skills, preferably with a Logo language, are necessary. Those with good NetLogo programming competence can be successful, but there will be challenges:

- Import process needs improvement
- NetLogo comments are not imported, thus removing potentially important code documentation
- NetLogo imports are limited in exposing ReLogo’s full potential
- ReLogo will always be at least somewhat more difficult to program than NetLogo

As with Repast for Java, Repast ReLogo lacks adequate documentation and example models, which steepens the learning curve.

*Score: C- (pending improvements in update releases)*

**Support for importing data and maps**

Text files can be read and written; all formats can be read with some custom programming.

Concerning GIS map files:
• Repast contains powerful custom Java API for ESRI Shapefile (Vector) import and processing of vector data and data attributes. Shapefile export utility exists, but has not been tested by us.

• Native ReLogo support for basic ASCII Grid (Raster GIS) import and mapping to patch agents (a la NetLogo); requires minimal programming.

• More comprehensive Raster GIS support is lacking; however, a remedy, possibly via adaptation of open-source GIS tools, might involve relatively modest programming effort.

In general, ReLogo’s built-in functionality with regard to data input of various kinds can be expanded with custom Java programming.

*Score: B-

**Documentation and support**

The Repast website (http://repast.sourceforge.net/) has been completely rewritten with the introduction of Repast Simphony 2.0 and Repast HPC. It is the essential starting point for Repast documentation and is well done.

The “Getting Started” series of guides, for ReLogo, Java, and Repast FlowChart Programming, new with Repast Simphony 2.0, give good, thorough introductions to Repast. The ReLogo Getting Started guide seems especially well done, however the lack of a troubleshooting guide is problematic when mistakes arise. The accompanying “ReLogo Primitives Quick Reference” is also good, but it was not complete at the time of this writing.

Otherwise, Repast documentation could be much better. Complete standard API documentation (javadoc-generated) for Repast, Groovy, and ReLogo is available, but often with little or no explanatory commentary. Other documentation is often sparse, rudimentary, and sometimes out-of-date (understandable, given the recent, beta, introduction of Repast 2).

Another weakness is in the area of available example models. Sample models are few, especially for Repast 2 beta (this may be improved with time).

A Repast-dedicated user forum mailing list (“repast-interest”) is a major support plus. The forum is quite active, and Repast developers are typically
quite responsive and helpful. (See http://repast.sourceforge.net/support.html for links to the forum mailing list subscription site and to the list archives.). Training courses are available annually, but the courses focus on business applications and would likely not be interesting to most ecologists.

Score: C (documentation)
   B (other support)

**Multi-processor and distributed computation support**

Repast Simphony includes a Parameter Sweep utility for management of multiple model runs with fine-grained parameter control. The utility can be used with open-source GridGain software for multiprocessor and cloud distribution of model runs (installed separately, http://www.gridgain.com).

Repast Simphony also includes built-in support for multi-thread action scheduling via the Repast Java ThreadedAction class. Repast for High Performance Computing includes an explicit framework and support for distributed processing of individual models.

Score: A-

**A Summary Note**

Repast Simphony is a powerful, well-developed software package that provides several language options for developing agent-based simulation models that could be used for ecological modeling. It has several strengths:

- Sophisticated, mature, extensive, and imaginative Java-based modeling system enables the creation of very ambitious and powerful models, for those with the proper skills.
- Full integration with the powerful Eclipse development environment.
- Support of relatively small but very strong research-oriented team of developers at Argonne National Laboratory.
- ReLogo is a new, relatively easy-to-use and exciting tool with great promise. It enables seamless integration with the entire Repast Java-based modeling system and custom programming tools.
- Repast Simphony could play a significant role as a component-authoring tool within a component-based modeling system.
However, it also has several major weaknesses:

- Lack of adequate documentation, and small number of sample models create a steep learning curve for burgeoning modelers. Documentation and models are satisfactory on an introductory level, but there is very little help with the more complex potential of Repast, including ReLogo.
- Lack of comprehensive Raster GIS implementation; current Repast raster GIS tools are insufficient to support this critical aspect of much ecological modeling.
- Ease of use is only fair, particularly in comparison with a platform like NetLogo. Again, in comparison with NetLogo, more, and often more difficult programming is needed to accomplish many modeling tasks well.
- Use as a center or framework for a component-based modeling system is not part of Repast’s design.

**Summary Scores: Repast Simphony and ReLogo**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strength of system, on-going development</td>
<td>A-</td>
</tr>
<tr>
<td>Support for Spatially explicit modeling</td>
<td>A-</td>
</tr>
<tr>
<td>Library of existing models supporting ecological analyses</td>
<td>C</td>
</tr>
<tr>
<td>Opportunities for connecting to other models</td>
<td>A-</td>
</tr>
<tr>
<td>Support for component-based modeling</td>
<td>D/B (as a framework/component dev.)</td>
</tr>
<tr>
<td>Platform independence</td>
<td>A</td>
</tr>
<tr>
<td>Connections to external data analysis tools</td>
<td>B-</td>
</tr>
<tr>
<td>Easy entry for non-programmers?</td>
<td>C- (pending improvement)</td>
</tr>
<tr>
<td>Support for importing data and maps</td>
<td>B-</td>
</tr>
<tr>
<td>Documentation and support</td>
<td>C-/B (doc./support)</td>
</tr>
<tr>
<td>Multi-processor and distributed computation support</td>
<td>A-</td>
</tr>
</tbody>
</table>
Appendix D: Netlogo

Author: James Westervelt

Use of Netlogo by ERDC-CERL

In 2008 a small group of ERDC researchers began tackling the challenge of the ecologist-programmer chasm. In conjunction with Professor Bruce Hannon at the University of Illinois, ERDC’s Jim Westervelt had established a course to teach spatially explicit ecological modeling beginning in the mid 1990s and that course had used the non-spatial, but non-programmer friendly Stella modeling software in conjunction with a powerful but non-programmer unfriendly capability called the Spatially Modeling Environment (SME). SME was able to ingest non-spatial Stella models and generate a C++ code version of the Stella model that could be run simultaneously within every grid cell of a raster-GIS area of interest. Students received experience in the design and development of spatially explicit models through the development of Stella models that were processed by a highly trained post-doctoral student running SME, but were unable to leave the class with the ability to take the resulting model or develop a model on their own. In 2008 this problem was solved after a review of available software when Westervelt and Hannon agreed to adopt Netlogo for the Spring 2008 offering of the class. Several classes of students and ERDC-CERL ecologists have since learned to write spatially explicit models using Netlogo. Netlogo was also independently discovered and adopted by an ERDC-CERL social-modeling group. The number of ERDC researchers, students, and contractors that are currently familiar with Netlogo programming include:

- Chandler Armstrong
- Jen Burton
- Michael Case
- Marina Drigo
- Patrick Edwards
- Charles Ehlschlaeger
- Jay Johnson
- Rick Lance
- Kirk McGraw
- Bill Meyer
- Natalie Myers
- Paul Pawlenko
- Jinelle Sperry
- David Stigberg
- Jim Westervelt
In recent years a wide variety of Netlogo-based models have been developed and documented including:

- Fort Benning Gopher Tortoise
- Patch Valuation Model
- Fort Stewart Gopher Tortoise
- Fort Stewart Striped Newt
- Fort Stewart Flatwoods Salamander
- Unmanned Aerial Vehicle Routing in 3-D Space
- Relationships among Forces in Afghanistan
- Food Security in a South American City
- Domestic Violence in Chicago
- Fort Benning Feral Hogs
- Fort Hood Fire Ant Predation on Cave Crickets
- Nutrient Cycling in the Mississippi
- Urban Growth
- Metapopulation Migration Simulation

System introduction

In 1967 Seymore Papert (image at right) created the Logo programming language for educational use. It provided a math land that allowed grade school kids to play with words and sentences and was used to program a turtle robot. The language construct “ask turtles [move forward ]” reflects that the original use of the Logo language was to ask the robot turtle to do things. With the advent of the Apple MacIntosh in 1984, the Logo language became accessible to many more people and was used to give instructions to virtual turtles and to the tile floor in a virtual on-screen world.

In an educational setting it is always best to base instruction on what the students bring to the process. The Logo language does this by being innately easy to read. Consider:

```
ask turtles [ move forward ]
```

The person new to the Logo language would assume that there are things (more than one) in a little world that can be thought of as separate entities
that have the ability to move in that world and that they are being asked to “move forward”. This provides an easy and intuitive entry into the language that suggests to the student that perhaps computer programming could be easy after all. Now consider:

```
ask turtles [ left 90 move forward ]
```

Here the turtles are each given two instructions and those instructions are contained within square braces, which suggests that perhaps an unlimited of instructions might be given to the turtles. The first instruction asks each turtle to turn left 90 degrees. With such an introduction, a person new to Logo is poised to embark on an empowering adventure that will unveil a nearly unlimited language. Remember always that this language was first developed to be accessible to grade school children, with nearly unlimited possibilities. Like a human language, it is easy to begin to learn yet can take years to master.

The Logo language has been implemented over the years in nearly 200 versions including UCBLogo out of Berkeley, Starlogo out of MIT, Netlogo out of Northwestern University, and ReLogo out of Argonne National Laboratory. We focus here on Netlogo, developed under the direction of Uri Wilenski who studied as a doctoral student under Seymore Papert.

Netlogo is fundamentally an agent-based spatially explicit simulation model design and development environment. Agents take the form of a landscape of regularly spaced grid cells called patches and moveable entities. By default, a breed of moveable agents called turtles can be instantiated (created) and imbued with the power to respond to native and programmed commands. However, any number of user defined “breeds” can be established, each with its own unique responses to commands that guide them through the virtual world replete with dynamic landscape patches and other moveable agents.

Netlogo has become popular within ERDC and has been used in support of ecological, social, and military modeling efforts. Eleven models are documented in a recent ERDC book published by Springer (Westervelt 2012).
The Netlogo interface

The initial Netlogo experience is via a graphical user interface that provides tabs allowing access to three aspects: Interface, Information, and Procedures. The interface tab provides the end-user experience involving various inputs and outputs (Figure 3) that include buttons, sliders, labels, graphs, readouts, and a dynamic map. The Netlogo programmer creates and positions these as desired. This example is of a salamander model. The background of the map is a shaded-relief image over which vector GIS maps showing roads (black) and streams (blue) have been placed. The green dots indicate locations of individual animals. The second tab, information, provides access to simple text that typically explains how to run the model. The third tab opens the actual Netlogo program.

Figure 5. Netlogo interface example
**Strength of system, on-going development/support**

Netlogo has been actively released since 1999 and is associated with an active community. It is Java-based, meaning that it runs on any Windows, Linux, or Apple computer running Java. Free downloads are available for any of these platforms. Netlogo is very extensible and a significant number of third-party extensions are available. The majority of Netlogo is open source, and the remainder can be made available through non-disclosure agreements. Dr. Uri Wilenski of Northwestern University directs the Netlogo development and continues to have many students involved in further developing and applying the software, which results in a very active core community.

**Support for spatially explicit dynamic eco-modeling**

Netlogo is fundamentally a spatially explicit 2D agent-based simulation-modeling environment. Netlogo is also available as a 3D system where 2D patches are replaced with 3D voxels, which can support the development of models that specify behaviors of agents through a water or air space. The basic agents are patches, square patches of “land”, and moveable entities that can be established as breeds. Most Netlogo models embrace the notion of time, yet Netlogo does not enforce the idea of a defined time step. It provides a type of clock, but ticks on that clock are under the control of the modeler. The programmer typically establishes a time step as everything that occurs when a particular procedure (conventionally called “Go”) completes. Time, if modeled at all, typically passes as a standard amount of time, such as a day, a week, a month, etc.

Netlogo captures the surface of the modeling playground as a regular array of square grids (patches) that represent a dimension of the user’s choosing. The extent of this space and size of the patches is established up front and fixed. It is possible for a modeler to ignore the Netlogo patches and instead establish a different arrangement of patches (e.g., hexagons) by using a new breed.

The community of raster-based GIS users find that Netlogo’s default square patches are familiar and, indeed, can be initialized using raster GIS maps. The state of a Netlogo model can also be saved during or after simulations as raster GIS maps.
What ecological hierarchies does Netlogo support? Netlogo is not specifically an ecological modeling environment. It comes with a wide variety of sample models that collectively demonstrate the broad range of application domains to which it has been applied, including art, biology, populations, communities, chemistry, physics, sociology, earth science, and climate change. The notion of mobile agents in a 2D world makes it especially applicable to landscape systems within which individuals interact with one another and with surfaces that change and behave over time.
Opportunities for connecting to other models

Being Java-based, Netlogo models can be embedded in and controlled by Java programs. There is a complete and documented application programmer interface that gives one access to the Netlogo software.

NetLogo has an extension facility that allows its capabilities to be extended from Java (and other JVM languages). Authors of extensions are encouraged to share their extensions with the community. Netlogo comes with a set of bundled extension that include capabilities that support matrices, arrays, tables, GoGo electronic hardware, a profiler, sound, GIS, and others. Users currently can download other extensions to extend the Netlogo capabilities:

- BDI and FIPA ACL: Supports development of goal-oriented agents that communicate using FIPA-ACL messages
- MIDI: A more complete support for MIDI than the sound extension
- SynthExtension: To output sound from NetLogo without using files or MIDI.
- Nlboris: Enables Netlogo agents to send messages to/from other NetLogo agents
- Multiview: Adds further windows for displaying patches.
- Pathdir: An extension for finding the current working, user and model directories; creating, moving, renaming, identifying and deleting directories; and listing the contents of directories.
- Linear Programming Solver: Calls LPsolve55 mixed-integer linear programming solver from within a Netlogo model.
- MySQL: For accessing MySQL databases and executing SQL commands such as select, insert, update, etc.
- Random Number Generators: Includes all of the core Netlogo probability distributions, as well as Negative Binomial and Beta distributions
- Shell: Invoke external shell commands, read and set environment variables.
- jGE (Java Grammatical Evolution): Allows NetLogo users to get familiar with and use Grammatical Evolution within their models.

Support for component-based modeling

Netlogo is not designed to be a component-based modeling environment. There is no sense of the ability to develop separate modeled aspects of a
system as components and then arbitrarily link them together. However, with certain conventions, some level of component modeling is possible. And, as of the summer of 2011, two doctoral students under the direction of Uri Wilenski were exploring the development of an umbrella over Netlogo that would allow Netlogo programs to run simultaneously as components of a larger system. Also, in the summer of 2011 Object Modeling System programmers successfully demonstrated and documented the capturing of a simple Netlogo model as an OMS object. Nevertheless, Netlogo is fundamentally unsuited as a component-based modeling system.

Platform independence

Netlogo is in active development and releases continue to make use of the latest Java capabilities. As such, it runs on all modern computers running the latest versions of Java including Windows, Mac OSX, and Linux.

Netlogo supports multi-threading on a single platform for running simultaneous instances of the program, but has no capability to run a single Netlogo model across multiple local or distributed processors. Similarly, there is no supercomputer support. Ecological simulation modeling typically contains substantial stochasticity, which results in the requirement for hundreds or thousands of runs. The ability for Netlogo to run these simultaneously as multiple threads is a perfect way to utilize multiple processors on a single platform. Through batch scripts it is possible to start simulations on other computers; even across an array of machines that are part of a supercomputer.

Connections to external data analysis tools

Netlogo has some extended ability to read and write data files. Natively, it can read/write ASCII information, and via extensions it can read/write some forms of GIS maps (ArcGIS shapefiles and Arc ASCII grid files), generate mpg movies, and read/write information to MySQL data files. With the ability to run shell scripts, Netlogo can facilitate the writing of information that is then, during a simulation, processed with external batch-based programs that generate results that can be read back into the Netlogo program.
Easy entry for non-programmers

Netlogo is one of the most approachable programming environments for non-programmers. The vast library of sample models provides many short and simple examples that allows a new person to entertainingly explore the extent of the Netlogo capabilities. The model code is built into and accessible from each model where the Netlogo explorer is free to see how the model works and to immediately begin altering the model. By moving between the model interface and the model code, one can get instant feedback on the consequences of code modifications. It is not uncommon to see a non-programmer developing completely new models within days or weeks of their Netlogo experience.

Unlimited opportunities for programmers

On the surface, Netlogo appears to be a “toy” programming environment. This surface is simply the general public aspect that appropriately invites new modelers into this world. Beneath the surface lies an extensive world that provides almost limitless opportunities.

The advanced Netlogo modeler can create libraries of reusable Netlogo code in the form of .nls files. While not exactly components, such code can add procedures available within a new model. ERDC developers, for example, have put genetic inbreeding calculations into a .nls file that can be used for any species that might be the object of a new model. Another group has created a suite of procedures that makes Netlogo more object-oriented.

Java programmers can extend the Netlogo language by creating extensions that can be invoked from within a Netlogo model. Many such extensions are listed above in “Opportunities for connecting to other models.” Java programmers can also embed Netlogo models within Java programs allowing the model to be controlled and potentially linked with other simulation capabilities.

Documentation

The Netlogo 400+ page user manual is excellent as is its default extensive array of sample models. Additionally, many more models, extensions, tutorials, and how-to instructions can be found on the Internet. The active
Netlogo community provides a wealth of expertise upon which one can solicit advice, instruction, and guidance.

**What else?**

Netlogo provides a couple of capabilities that encourage participation and technology transfer. A group of individuals can collaborative control agents within a model through computer-computer communications facilitated by Netlogo’s Hubnet. This is useful for gaming situations where human decisions can influence the realization of alternate courses of action.

Netlogo models can run as Applets within Web browsers. This provides an easy way to share a model with potential users without the challenges associated with requiring users to download and install software. This approach is limited to models that do not read/write files or make use of extensions.

**The Netlogo language**

The Netlogo programming language provides commands to many internal procedures, grouped as follows to provide a flavor of the language’s extent. An extensive 400+ page programmer manual provides an excellent description of this entire language.

**Turtle-related**

back (bk) <breeds>-at <breeds>-here <breeds>-on can-move? clear-turtles (ct) create-<breeds> create-ordered-<breeds> create-ordered-turtles (cro) create-turtles (crt) die distance distance xy downhill downhill4 dx dy face face xy forward (fd) hatch hatch-<breeds> hide-turtle (ht) home inspect is-<breed>? is-turtle? jump layout-circle left (lt) move-to myself nobody no-turtles of other patch-ahead patch-at patch-at-heading-and-distance patch-here patch-left-and-ahead patch-right-and-ahead pen-down (pd) pen-erase (pe) pen-up (pu) random-xcor random-ycor right (rt) self set-default-shape __set-line-thickness setxy shapes show-turtle (st) sprout sprout-<breeds> stamp stamp-erase subject subtract-headings tie towards towards xy turtle turtle-set turtles turtles-at turtles-here turtles-on turtles-own untie uphill uphill4
Patch-related

clear-patches (cp) diffuse diffuse4 distance distancexy import-pcolors
import-pcolors-rgb inspect is-patch? myself neighbors neighbors4 nobody
patches patches-own random-pxcor random-pycor self sprout sprout-
<breeds> subject turtles-here

Agentset

all? any? ask ask-concurrent at-points <breeds>-at <breeds>-here
<breeds>-on count in-cone in-radius is-agent? is-agentset? is-patch-set?
is-turtle-set? link-heading link-length link-set link-shapes max-n-of max-one-of min-n-of min-one-of n-of neighbors neighbors4 no-patches no-
turtles of one-of other patch-set patches sort sort-by turtle-set turtles with
with-max with-min turtles-at turtles-here turtles-on

Color

approximate-hsb approximate-rgb base-colors color extract-hsb extract-
rgb hsb import-pcolors import-pcolors-rgb pcolor rgb scale-color shade-
of? wrap-color

Control flow and logic

and ask ask-concurrent carefully end error-message every foreach if ifelse
ifelse-value let loop map not or repeat report run runresult ; (semicolon)
set stop startup to to-report wait while with-local-randomness without-
interruption xor

World

clear-all (ca) clear-drawing (cd) clear-patches (cp) clear-turtles (ct) display
import-drawing import-pcolors import-pcolors-rgb no-display max-pxcor
max-pycor min-pxcor min-pycor patch-size reset-ticks resize-world set-
patch-size tick tick-advance ticks world-width world-height
**Perspective**

follow follow-me reset-perspective (rp) ride ride-me subject watch watch-me

**HubNet**


**Input/output**


**File**


**List**

but-first but-last empty? filter first foreach fput histogram is-list? item last length list lput map member? modes n-of n-values of position one-of reduce remove remove-duplicates remove-item replace-item reverse sentence shuffle sort sort-by sublist
String

Operators (<, >, =, !=, <=, >=) but-first but-last empty? first is-string? item last length member? position remove remove-item read-from-string replace-item reverse substring word

Mathematical

Arithmetic Operators (+, *, -, /, ^, <, >, =, !=, <=, >=) abs acos asin atan ceiling cos e exp floor int is-number? ln log max mean median min mod modes new-seed pi precision random random-exponential random-float random-gamma random-normal random-poisson random-seed remainder round sin sqrt standard-deviation subtract-headings sum tan variance

Plotting


Links

Movie

movie-cancel movie-close movie-grab-view movie-grab-interface movie-set-frame-rate movie-start movie-status

System

netlogo-applet? netlogo-version

Built-In Turtle Variables

breed color heading hidden? label label-color pen-mode pen-size shape size who xcor ycor

Built-In Patch Variables

pcolor plabel plabel-color pxcor pycor

Built-In Link Variables

breed color end1 end2 hidden? label label-color shape thickness tie-mode

Keywords

breed directed-link-breed end extensions globals __includes patches-own to to-report turtles-own undirected-link-breed

Summary Discussion

The overall “grades” given in this review for the various categories is as follows:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Strength of system, on-going development/support</td>
<td>A</td>
</tr>
<tr>
<td>Support for spatially explicit dynamic eco-modeling</td>
<td>A</td>
</tr>
<tr>
<td>Library of existing models supporting ecological analyses</td>
<td>A</td>
</tr>
<tr>
<td>Opportunities for connecting to other models</td>
<td>B</td>
</tr>
<tr>
<td>Support for component-based modeling</td>
<td>B</td>
</tr>
<tr>
<td>Platform independence</td>
<td>B+</td>
</tr>
<tr>
<td>Connections to external data analysis tools</td>
<td>A</td>
</tr>
</tbody>
</table>
The adoption of Netlogo has, and will continue, to address many of the EMS goals developed through the two EMS workshops in 2011. It has a very low threshold for adoption by non-programmer ecologists. It makes use of the Logo programming language, which was originally developed for grade school children. That original language has developed into a full-featured and powerful open-ended high-level language. Modelers can focus on the dynamics of their particular model and use the built-in capabilities of Netlogo to provide such things as user-interface graphics, output graphics, file reading/writing, GIS input/output, and compilation. New modelers find Netlogo to be easy to learn, while old hats continue to find virtually endless modeling possibilities.
Appendix E: Object Modeling System (OMS)
3.1rc3

Authors: Billy Johnson and Charles Ehlschlaeger

OMS Introduction

The Open Modeling System (OMS) is designed to be a general purpose framework for integrating models together. Its primary strength is its ease to facilitate component or model interoperability across programming languages, modeling frameworks, and architectures. The latest version, 3.1rc3, is used similar to a scripting language like UNIX shell scripts or PERL: The modeler can write ASCII scripts that define which components connect to other components and how to execute those components. But, when OMS executes the new construct, it will run at compiled speeds whenever possible. For example, two Fortran based components passing doubles to each other will run as if it were a single Fortran program. Object oriented components can pass objects to each other at compiled speeds as well.

OMS has been in development since 2001 with regular releases since 2005. Much of the technology has been built from efforts by the United States Department of Agriculture and Colorado State University. OMS has several international collaborators and is being used by the uDig open source geographic information system for running JGrass and Grass commands. Current OMS applications are dataflow driven, allowing large amounts of data from many formats to be passed among many different applications.

OMS as an open source platform is developed by Dr. Olaf David, who is responsible for core design as well as software releases. Dr. Olaf David is currently a research scientist at Colorado State University at Civil and Environmental Engineering and Computer Science Department working in collaboration with OMSLab, a just founded CSU institution to advance OMS and facilitate research partnerships. Currently, Colorado State University and USDA are sponsoring the project. OMS is copyrighted with a LGPL license and is available at no cost from http://oms.javaforg.com.
Support for spatially explicit dynamic ecological modeling

While OMS doesn’t provide spatial modeling tools, it eases the integration of independently designed and implemented software systems. Version 3.1 release candidate 3 directly supports NetLogo models, which has a long history of spatially explicit dynamic ecological modeling. OMS also directly supports JGrassTools, which provides a complete set of modules dedicated to hydro-geomorphology and treats all non interactive GRASS commands as components. Ecological models often have statistical components in them. To facilitate those models, OMS supports the statistical package R, allowing R commands to act as components. If the modeler needs capabilities not found in JGrassTools, NetLogo, or other existing OMS components: Java, C, C++, or Fortran code can be converted into OMS components in a matter of hours by a programmer who understands OMS, the appropriate programming language, and the needed capabilities.

OMS has no dimensional, temporal, or scale limitations. It is limited only by the constraints of the programming environment chosen for its components. For example, components written in Java will have a memory limitation determined by the specific runtime engine installed on the computer(s) running the model. Models can be one dimensional, 1D with time, 2D, 2D + T, 3D, 3D spherical, etc. Currently hydrologic, agro-environmental, forecasting, isotope, groundwater, erosion, and ecosystem-based models exist using OMS.

A spatially explicit dynamic ecological modeling environment using OMS today

As mentioned in the previous two sections, OMS is built into uDig as uDig’s “Spatial Toolbox” and OMS can integrate ecological models. Ergo, uDig can provide a 2D plus time spatially explicit dynamic ecological modeling environment today\(^1\). All of the non-interactive native GIS operations from GRASS are available from the Spatial Toolbox as well as all constructed OMS components.

\(^1\) This section assumes that OMS version 3.1 release candidate 3 can be used as the Spatial Toolbox. By the time ERDC implements the “Next Generation Ecological Modeling System,” uDig will definitely have NetLogo functionality built into its Spatial Toolbox. At this time, copying oms-all.jar from oms.javaforge.com->oms-3.1rc3-console.zip to uDig’s $uDigHOME/plugins/net.refractions.udig.libs_1.2.2.201107130220/lib folder and overwriting the file jgt-oms3.0.7.1.jar will allow properly built NetLogo OMS components to be run from within uDig. OMSLab will have a more streamlined installation progress when the next full version of OMS, 3.2, is available.
Figure 1 shows uDig after running a DEM gradient OMS component. While the modeling user interface (lower right in figure) is primitive, OMS components written in Java, C, or NetLogo would allow for a more polished WIMP (window, icon, menu, and pointing device) experience. Overall, uDig provides a GIS user interface with various quirks: someone not familiar with uDig will sometimes not understand an intuitive way for achieving a specific cartographic effect. (This is especially true for people expecting uDig to be a clone of ArcMap.)

**Library for spatially explicit dynamic ecological modeling**

OMS provides a ready built file structure for storing and organizing ecological models. Assuming a user interface similar to uDig, users would be able to open libraries of models with the Modules section of the Spatial Toolbox (see Figure 1). OMS allows components to be enabled with many metadata tags useful for advanced searches. uDig’s Search tab currently
only provides data searches but could easily extended to include OMS model metadata.

**Support for component-based modeling**

OMS was originally designed to provide component-based modeling support. It supports location-specific model development using a high-level language. OMS can encapsulate existing ERDC models quickly. A programmer familiar with an ERDC model and a programmer familiar with OMS could encapsulate the ERDC model in hours or days. OMS supports full dynamic feedback among model components. Some analytical tools exist to automatically determine whether a model is well constructed or not. For example, if the links are not connected such that specific output branches are reached, OMS can flag the model as incomplete. OMS allows the quick addition of ad-hoc model components for modelers familiar with scripting languages. One of OMS's greatest strengths is its metadata support. It can reveal metadata about components to help modelers best select and connect components.

OMS fully promotes component-based model development. If a component is written in a 3rd generation programming language, Java, Fortran, or C/C++, OMS links components allowing binary reuse. In other words, complementary components execute the model at speeds equivalent to a single monolithic program. 3rd generation programming components retain the source code in the original language. Metadata in comments or in a separate file provides a noninvasive way to define the components. This component design process facilitates multi-language component integration without modifying the original source code, preventing the “forking” of source code. If an OMS component is written in a 4th generation programming language, such as NetLogo, object components are scanned at compile time, preserving OMS model design tools. However, the source code interpreter must first begin running before the code begins executing. It is likely that an OMS model of many NetLogo components will have a significant slowing due to the restarting of the NetLogo engine with each NetLogo component.

---

1 FORTRAN, C, Java, and similar are considered to be 3rd generation programming languages, while OMS would be considered a 4th generation programming language.

2 Olaf David expects the next version of OMS to only instantiate the NetLogo engine once. Once achieved, componentized NetLogo programs should run as quickly in OMS as the original programs.
OMS currently contains about 250+ modeling components, enabling Monte Carlo simulation, ensemble stream flow prediction ESP, shuffled complex evolution SCE, FAST sensitivity analysis, and LUCA (USGS), for example. OMS provides “one click” auto-documentation, and auto-testing, of components and models. OMS integrates into various IDEs for development including Netbeans, Eclipse, and IntelliJ. All ongoing USDA / CSU model development is open source and available under LGPL. Thus, USDA Hydrology, rainfall-runoff, erosion, sediment transport, phosphorus, nitrogen, range-livestock, isotope, and distributed watershed models are available for use.

Large scale models (with +10,000 or more lines of source code) as well as smaller educational examples are available to assist development.

**Platform Independence**

OMS runs on any computer with Java installed including Android phones and tablets. There are single downloadable install files for Windows, Mac OSX, and Linux. OMS supports Linux based cloud or cluster computing.
Connections to external data analysis tools

Data analysis tools, which can run the gauntlet from visualization to statistical analysis, can be connected between software packages in three major ways: 1) require the user to manually select files from the operating system, 2) have separate software program share standardized files between them, and 3) have separate software programs have deep connections allowing binary data to pipe between them without invoking operating system functions.

OMS allows deep connections (3) between all 3rd generation language software programs that have been defined as OMS objects. Thus, most of the USDA OMS models have these deep connections. JGrasstools and Worldwind (Geowind) also has deep connections. Data analysis tools such as uDig and R can easily share standardized files with OMS components. NetLogo models can connect with other OMS components through shared standardized files. (However, it would be trivial for professional programmers to construct extensions in NetLogo that would convert data structures into Java classes which could be “connected deeply” with Java based OMS components.) Constructing an OMS component to read or write an instance of a Java class should take less than a day by an experienced programmer.

The most common standard GIS map formats, ESRI shapefiles for vector objects, ESRI ASCII grids, and GRASS raster files, can be read into various OMS components. Excel spreadsheets can also be read into various OMS components. “tables”, which are defined as comma separated values files with metadata, can be read by the OMS3 DataIO API.

Easy entry for non-programming modelers

While programmers should be able to implement OMS components within days, non-programmers who understand simulation modeling concepts will need at least a week of training and practice before they can start being “productive” in an OMS environment. To be minimally productive, a modeler would need to study the set of existing OMS components, learn to setup and use simulation scripts, and understand the final user interface. Currently, OMS 3.1 can only integrate components using simulation scripts. Olaf David estimates that the first version of an OMS Graphical Model Builder to integrate components will be completed by the National Water and Climate Center (Portland) in 2012. Depending on the quality of
OMS Graphical Model Builder and how well it integrates with OMS documentation tools, the OMS Graphical Model Builder should reduce the training time for non-programmers.

Any modeler will be able to exploit auto-documentation tools, and creating simulation audit trails that will speed model development.

**Use by programmers**

Programmers, with as little as several days of training, can develop OMS components from Java, FORTRAN, C, or C++ code by annotating component inputs, outputs, and executables. OMS components can also be developed for NetLogo models. All annotation-based components are non-invasive and developed with a simple methodology that can be understood by even novice programmers.

OMS currently requires model developers to construct model scripts. The OMS scripting language will look familiar to anyone who has built object-oriented classes. Figure 3 indicates the required inputs and outputs of the JGrassTool Aspect.

![Figure 9. Components of JGrassTool Aspect](image)

The following OMS script assigns default values to various parameters of the JGrassTool Aspect as it is being defined as an OMS component.
A more complex OMS script example is below. This script uses six OMS components: pit, flow, slope, rasterreader, rasterwriter, and mapsviewer to display the resulting slope map in a map viewer. The process is as follows (in rough chronological order):

1. The digital elevation map “dem” located in
   /data/spearfish/PERMANENT/cell is passed to a raster reader.
2. The raster reader, rasterreader1, output is directly connected to a pit calculator component input.
3. The pit component output is directly connected to a flow component input.
4. Both the pit component and flow component outputs are connected to a slope component’s inputs.
5. The slope component’s output is connected to the map viewer.
6. All outputs are passed to raster writers to save the results.

Since OMS allows distributed processing, the processes need not occur in the exact order described. Raster writers may be processed before, after, or during the map viewer. Since the slope component requires both a flow map and a pit map, those components will have to run beforehand.
Dr. Olaf David and other USDA representatives have encouraged ERDC, should we adopt OMS in our research, to contribute to OMS itself. ERDC can contribute in both software development, future development of OMS design, or even provide members to an OMS Board of Trustees. It is easy to join the OMS project at oms.javaforge.com as a developer. There is also an avenue for ERDC to become board member of OMSLab and
actively steer the direction for future development, collaboration, and outreach.

**Documentation**

Documentation is currently over 150 pages long, but it is still in draft form. There are example projects with accompanying data sets and an FAQ. These are available from oms.javaforge.com. OMSLab is preparing a 3-4 day training course for up to 12 trainees to be taught by Dr. Olaf David. On several occasions, Dr. Olaf David has provided ad hoc training with ERDC personnel that brought ERDC source code. Dr. Olaf David taught programmers how to implement OMS on the provided source code.

**Multi-processor and distributed computation support**

OMS supports high-performance computing. As mentioned earlier, OMS is dataflow driven. Each component is given its own thread on a multi-threaded system. OMS uses Hadoop MapReduce, [http://hadoop.apache.org/mapreduce/](http://hadoop.apache.org/mapreduce/), for applications that rapidly process vast amounts of data in parallel on large clusters of compute nodes.

The USDA has already developed cloud service applications as part of the five year research effort.

*Figure 10. Cloud computing allows model results on smart phones.*
**OMS Future**

To ERDC EMS’s benefit, ModelBuilder provides the largest short term improvement to OMS. ModelBuilder will provide visual component integration, runtime, and debugging with a scheduled completion date for 2012. ModelBuilder has already prototyped. Once routinely available, it should provide non-programmers a much easier component modeling experience.

USDA plans further focus on large scale computing using OMS3 with integration into USDA Cloud Services Innovation Platform in collaboration with Colorado State University’s Civil Engineering and CS departments. This research will leveraging models on a wide range of architectures, from Android to compute clouds.

OMSLab has been at CSU since May 2011. Its mission is to facilitate collaboration on OMS advancement, assist in collaborative modeling efforts, provide training, host short and long term visits of modelers and users, and conduct thesis projects with MSc and PhD students that are modeling framework related.

**Best Things for ERDC EMS**

Beyond the many issues discussed earlier, here are several miscellaneous benefits OMS has for current and future ERDC research. Multiple U.S. agencies have already bought into the OMS system: USDA, NRCS, ARS, and the USGS. The collaboration opportunities to share development costs are many. The OMS3 programming environment offers opportunities to construct future models more quickly than similar approaches. Lloyd et al. (2011) analyzed the impact of framework invasiveness to model design. Framework invasiveness can be thought of as the impact of the programming language and tools on how effective the software development is. Lloyd’s research indicates that the amount of source code and the complexity of the code for a specific ecological model were reduced by 10 - 15% by using annotated modeling components, specifically OMS 3.0. The research indicated that OMS 3.0 was the only component based system to use fewer lines of code than when writing the ecological model in the native 3rd generation programming language. (Of course, this assumes the developers are already familiar with OMS.) Overall, Lloyd claims the OMS 3.0 framework produced less invasive model implementations when compared to the heavyweight framework.
implementations using ESMF 3.1.1, OpenMI 1.4, and CCA 0.6.6. However, since the authors of that study were associated with Colorado State University, they might have biased in their assessment.

**Biggest concerns**

The two biggest concerns are ERDC cultural “buy in” and tools for training.

Developing numerical models within a framework such as OMS does require a shift in the way we approach model development. There has to be a “buy in” by ERDC to encourage its engineers and scientists to adopt this approach both in regards to modifying legacy code in addition to developing future code.

Also, training courses and tutorials for OMS are still in their infancy. Dr. Olaf David is currently the only person teaching OMS to groups. This problem will remain until a formal tutorial and training package is developed, shared, and used by multiple OMS developers for teaching purposes.
Appendix G: FRAMES

Author: James Westervelt

System Introduction

The Pacific Northwest National Laboratory originally developed the Framework for Risk Analysis in Multimedia Environmental Systems (FRAMES), which has now been used by a number of agencies including U.S. Environmental Protection Agency (EPA), U.S. Army Corps of Engineers, U.S. Nuclear Regulatory Commission, and the U.S. Department of Energy (DOE). Recently the EPA’s Ecosystems Research Division (ERD) in Athens has supported the ongoing development of FRAMES-2, released in 2005.

The scope, purpose, and philosophy behind FRAMES is similar to the European Union’s OpenMI. Its goal is to capture the investments in the design, development, and acceptance of legacy simulation models by facilitating the connection of these models to create more complete simulations of environmental systems. FRAMES is designed to support the “seamless transfer of data between disparate models, databases, and modeling systems.”

An ERD example involves the connection of 17 modules (modified legacy modules) representing chemical sources, transport, foodchain, and exposure/risk to create the 3MRA (Multi-media, Multi-pathway, Multi-receptor Risk Analysis) modeling system (Johnston et al. 2008; Zakikhani et al. 2006). An ERDC example is the ARAMS (Adaptive Risk Assessment Modeling System), which connects modules that simulate source zone, air, soil/vadose, aquifer, surface water, soil and overland runoff, human receptor intake, human health impacts, eco tissue concentration, eco receptor intake, human exposure pathways, and eco effects (Lloyd 2007).

To be useful in FRAMES, legacy models should be modified with FRAMES software that facilitates reading/writing FRAMES files that are used to communicate information among the models. Models thus converted into FRAMES-compliant modules then become part of a FRAMES component library. A FRAMES graphical user interface provides a module browsing,
selection, assembly, and execution environment that allows a modeler to link modules as needed for a specific application.

**The FRAMES Simulation Editor**

FRAMES distinguishes among model developers, database owners, and the end user/analyst. Model developers build or import DICs, define units, build/import modules, set up domains, and define connection schemes. FRAMES provides unit conversion, DICTionary, domain, module, and dataset editors. Database owners map database schemas, develop database extraction plans using FRAMES supplied data owner and data extraction tools. End users/analysts use simulation, data client, and dataset editors to select domains, select and connect modules using icons, select models and databases, and run data extraction plans.

**Figure 11. Sample FRAMES Simulation Editor**

Developed modules can be selected and assembled by modelers using the FRAMES user interface, and begin by selecting modules representing databases and models (left side of Figure 10) by dragging and dropping them into the model development environment on the right side of the figure. Arrowed lines are used to indicate connections from module to module. Icons in the upper right global workspace represent data that is
always available to all modules in the lower right workspace. Once assembly and parameterization is complete, the model can be operated through this interface and the traffic light icons are used to indicate model running, pausing, and stopping states.

Converting models to FRAMES modules

The exchange of information among modules is accomplished through precise well-defined files (Figure 11). These files have a known structure that the FRAMES model wrappering software libraries use and have precise definitions of the data in those files. The FRAMES community has developed a suite of DIC (dictionary) data types that contain data with agreed-upon metadata including units, definitions, type (float, integer, etc), and dimensions. Each wrappered model is designed to consume and produce data associated with specific DIC files. A given module can read data from one or more DIC files and similarly write out one or more files. As new model types become part of a FRAMES library.

**Figure 12. The FRAMES model integration philosophy (taken from a 2005 PNNL presentation)**

![Diagram of FRAMES model integration philosophy](image)

**Model wrapping in FRAMES-2 requires adding dictionary and description files, and model switches**

**Strength of system, on-going development/support**

FRAMES development has continued over the last 10 years, with PNNL boasting $8M in investments and $23M in multimedia modeling, which includes development of and applications with FRAMES. FRAMES 2.0 was made available in 2005. ARAMS was developed using FRAMES 1 and has not been modified to make use of 2.
Support for spatially explicit dynamic eco-modeling

Many of the models that have become FRAMES-compliant modules are spatially explicit. DIC files may store 1, 2, 3, or even 4 dimension data.

Opportunities for connecting to other models

FRAMES is all about component-based modeling, where those components are based on wrapping legacy models in code that moves information between the model and specific data files that allow passing that information to other FRAMES-compliant modules.

Platform independence

FRAMES runs in a Microsoft Windows operating system environment (Windows 2000 or XP), making use of MS Excel and the MS .NET framework.

Connections to external data analysis tools

Legacy models (see Figure 11) can import/export data into other files outside of the FRAMES DIC files. These may be in any form, including formats that allow for post-processing of model outputs. Alternately, information stored in the DIC files can be extracted, with software employing the FRAMES programming libraries, into formats that might be suitable for output post-processing.

Easy entry for non-programmers

Once a suite of models are converted into FRAMES-compliant modules, non-programmers can quickly and easily learn to plug the modules together to create application and location specific models.

Documentation

The primary FRAMES 2.0 starting point can be found at http://mepas.pnnl.gov/FramesV2. A web-based audio-video tutorial is available through that page and begins at http://mepas.pnnl.gov/genitutorial. This tutorial guides the viewer through the modeler experience of selecting and assembling modules into models that can then be run.
Summary discussion

FRAMES provides another solution to the challenge of adapting legacy stand-alone models of environmental systems and processes into an environment that allows these models to operate together and therefore more accurately capture a fuller set of a system’s dynamics. Like solutions offered by XMS, XMDF, and XDMF, the solution involves the careful crafting of files that FRAMES-wrapped legacy models read and write. The format of these files and the definition of the contained information is carefully crafted and controlled to avoid model-to-model miscommunication. The result is a powerful coarse-grained modeling system that a modeler can use to graphically link, parameterize, and run models based on accepted legacy models.
Appendix H: Component Based Modeling

Because of the complexity of natural ecological systems, the challenge faced in the management of any one system at any one location is likely to be different than the next management challenge. Therefore, ecological models are typically not reusable. Consider, for example a need for an oyster model. The model that gets developed can optimally focus on the management challenge and the decision space being considered by the oyster managers. Or, it can attempt to create an electronic version of a fully functioning oyster, which captures the functioning of organs and organ systems throughout time in response to internal and external conditions and resources. A “fully functioning” oyster model might be able to respond to:

- Food availability
- Water salinity
- Water toxicity
- Water clarity
- O₂ levels
- Predators
- Substrate availability
- Harvesting by fishermen
- Conditions needed by eggs, larvae, and adults
- Water currents
- Water temperature

If oyster reef managers are convinced that the current problem involves any one of these, then modeling the problem becomes much easier. If, for example, conditions are perfect for oyster development and the challenge involves over-harvesting, then a model that considers only oyster life-cycle and oyster harvesting patterns in time and space may be sufficient. This simpler model would not then be useful for situations where disease, predation, water quality, or substrate availability are key to the population densities. But, it is likely much more efficient to build and run even several problem-specific oyster models, rather than the one model that reproduces a “full functioning” oyster.
If ecological models are location, problem, and time-period specific, then it becomes necessary to build new models for each new ecological management challenge. Perhaps, pieces or components of models can be harvested and reused. If a model is built from easily separable parts, then it becomes possible to maintain libraries of these parts, or components. Consider the oyster example where a predator-prey oyster model exists, but we now need an oyster disease model. Components of the original model that might be useful in the new model might include data inputs, data input processing, data output visualization, data outputs, oyster egg and larvae stages, adult reproduction, and others.

Ecosystem components cannot be developed that are generically linkable to any other components. To link two components, the minimum requirement is that everything about the information provided by one component is exactly the same as the information needed by another. “Exactly” means that the following aspects of the information are identical:

- The precise definition of the data units
- The validity of the data over a time period
- The validity of the data over a spatial extent
- The precision of the data value

Therefore, suites of model components will be developed that will have components developed specifically for that suite. Each will be associated with a particular ecological management challenge and will be used to model the system at a particular ecological hierarchy and scale. Sample suites might include:

- Hydrologic
- Endangered bird species modeling
- Endangered reptile modeling
- Invasive species modeling
- Disease spread modeling
- Succession modeling
- Plant-plant resource competition modeling
- Soil chemistry and toxicology modeling
Data dictionary

The key to component-component linking begins with very specific data definitions that are associated with component inputs and outputs. Each will be defined by the following:

- Name
- Units
- Type
- Spatial resolution and extent
- Temporal resolution and extent
- Accuracy
- Format

Each data stream should have a unique name and unit combination. For example, “population of ants” in “individuals per square meter”, or “concentration of mercury” in “parts per billion”. The data type might be a single value, a vector of values, or a multidimensional array of values. The spatial resolution and extent refers to extent of the area covered by the value and the size of the grid cells or patches representing the data. This is equivalent to the header information associated with raster GIS maps that include coordinate system, projection, datum, corner coordinates, and number of rows and columns. The temporal resolution is the frequency at which the information is recalculated. Accuracy refers to the confidence in the values. Finally, format refers to the specific format in which the data is transferred. Sample formats include the many different GIS data formats, XML-based languages, the hierarchical data format (HDF 5), and others.

Input components

Spatially explicit simulation models should be initialized to some starting state, which is typically imported from tables, raster maps, and vector maps. Therefore, data import capabilities will form the basis of important reusable components. These components will be associated with the ability to read and convert many different formats, coordinate systems, datum, and projections.

Many pre-existing models have data requirements and have built-in software capabilities for reading input data. Componentizing these requires splitting out the self-reading of data in favor of generic data import components.
**Output components**

Similarly, generic data output components should be developed for converting model results into data formats that are readily consumed in any post-visualization or processing software steps. Like the input components, user selectable formats, coordinate systems, datum, and projections should be available. In addition to outputting tables, reports, and digital maps, output can be in forms to support post-modeling visualization such as presentation on websites or in systems such as Google Earth.

**Visualization components**

Visualization components are similar in nature to output components, but instead of saving system state information to files for later analysis and consumption; the information is displayed during a simulation run. Visualization can be used to help debug and develop a simulation or to provide visualization to decision makers using the model. Visualization modules can take forms such as:

- Run-time Messages
- Tables
- Graphs
- Video

**Analysis components**

Simulation models involve changes in system state that can be recorded for later analysis. Optionally, analysis of the system may be accomplished during simulation runs, which will require specialized components. Categories of desired analyses might include:

- Risk Analysis
- Statistical Analysis
- GIS Analysis

**System state components**

A system state component can be used to connect all other components together – avoiding the need for each component to “know about” and connect to each other for system state exchange purposes. Such a component plays the role of a central station or hub without providing any
system state change or simulation capabilities. It may simultaneously provide data translation services.

**Data translation components**

These components can allow for the connection of input requirements and output availability that are identical in definition, but don’t quite match in units or time/space resolution. Examples include converting to and from SI (International System) units or changing temporal or spatial resolutions. Using a data translation component alleviates the need to modify model code.

**Ecological process components**

There are two fundamental approaches in the development of potentially reusable ecological modeling components: processes and agents. Process modeling focuses on the some particular interaction, such as predator-prey (e.g. Lotka-Volterra), disease transmission, population exchanges across a meta-population, age-structure population growth (e.g. Leslie matrix), and habitat suitability index modeling. These are typically developed in a species-independent way that allows the model to be tailored to any particular species.

The downside of this approach is in its relative superficiality. There are many nuances to ecological systems that involve multiply overlaid networks that involve food, disease, relationships, memory, landscape structure, and communication. The later can take the form of postures, chemical signals, sounds, language, facial gestures, and light. The relatively simple process models typically ignore these.

**Ecological agent components**

Ecological agent modeling is conceptually easy to grasp as it directly reflects a common view of nature. That is, natural systems are composed of species (represented in software as agent classes), which appear in reality as individuals (software instantiated agents). The software class descriptions include descriptions of how individuals of the species behave, grow, and respond to stimuli and interactions. Individual agents are associated with specific information generally associated with individuals, such as age, health, location, sex, and hunger. The location places the
individual in the environment, which can point to external factors such as food, disease, shelter, water, neighbors, etc.

The downside of designing and developing species-specific agents is that the more specific the agent, the less likely the resulting model will be reusable. Agent modeling typically requires a tight connection between the possible combinations of external and internal system states and appropriate behavior that will respond to those combinations. Consider a two-species animal model that involves a predator-prey relationship. The model should appropriately capture the ability of both species to interact with one another and with a particular landscape that provides food, hiding, and shelter. Adding new agent classes (i.e., species) into this already working model can require extensive reworking of the original model. The new species may be a competitor of the predator species, may be a prey item of one or both species, or may be a predator of the original predator. It typically becomes necessary to reprogram the original species to appropriately respond to the presence of the new species.

One may believe that it should be possible to simply build a relatively complete model of a generic animal that is associated with a large enough array of traits and behaviors, so that individuals might be able to react to other individuals based completely on the traits of both individuals. For example, animal A is large, hungry and carnivorous, while B is small and herbivorous. In such a case A eats B, right? Unfortunately, the complexity in ecology results in what appears to be many exceptions. B may be able to avoid being eaten if it is well camouflaged, is toxic to A, can out run A, can out jump A, is aggressive and can inject poison, provides A with something more valuable than food, or appears sick.

**Greater system components**

The ecological system operates through interaction with the physical world, which itself can be very dynamic. The Corps of Engineers has substantial modeling experience in the field of hydrology – especially in the area of predicting the consequences of storms and weather on the nation’s aquatic systems. Therefore, the ecological modeling system should provide effective and appropriate approaches for linking hydrologic simulation models to ecological processes. This is expected to be especially true in the areas of important saltwater systems and wetland systems associated with the maintenance of freshwater shipping routes.
**REPORT DOCUMENTATION PAGE**

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

<table>
<thead>
<tr>
<th>1. REPORT DATE (DD-MM-YYYY)</th>
<th>2. REPORT TYPE</th>
<th>3. DATES COVERED (From - To)</th>
</tr>
</thead>
<tbody>
<tr>
<td>November 2014</td>
<td>Final</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4. TITLE AND SUBTITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ecological Modeling System: Conceptual Development and Design Specification</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5a. CONTRACT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5b. GRANT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5c. PROGRAM ELEMENT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5d. PROJECT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5e. TASK NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5f. WORK UNIT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6. AUTHOR(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Todd Swannack, James Westervelt, Jeff Hensley, and Robert Kennedy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environmental Laboratory</td>
</tr>
<tr>
<td>U.S. Army Engineer Research and Development Center</td>
</tr>
<tr>
<td>3909 Halls Ferry Road</td>
</tr>
<tr>
<td>Vicksburg, MS 39180-6199</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8. PERFORMING ORGANIZATION REPORT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERDC TR-14-11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)</th>
</tr>
</thead>
<tbody>
<tr>
<td>U.S. Army Corps of Engineers</td>
</tr>
<tr>
<td>Washington, DC 20314-1000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>10. SPONSOR/MONITOR’S ACRONYM(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>11. SPONSOR/MONITOR’S REPORT NUMBER(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>12. DISTRIBUTION / AVAILABILITY STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approved for public release; distribution is unlimited.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>13. SUPPLEMENTARY NOTES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>14. ABSTRACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>As military installations face continued ecological changes caused by changing land use patterns, changing climate, invasive species, and changing missions, the need to forecast the implication of such changes on habitats and at-risk species continues to grow. This requires ecological simulation modeling approaches based on ecological knowledge of the interaction of ecological components at varying time and space scales, at varying ecological hierarchies, and mediated through numerous relationship networks. Because each location and system challenge is unique, a library-based ecological modeling system is required to create cost-effective simulation models. A general purpose, cross-ERDC, widely-adopted ecological modeling system would provide a platform that allows model component reuse, multidisciplinary collaboration, and the ability to rapidly respond to emerging modeling challenges. This report documents the development of a plan for the creation of an ERDC Ecological Modeling System (EMS) based on the challenges and needs expressed through two ERDC-wide workshops held in 2011.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15. SUBJECT TERMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ecological Modeling</td>
</tr>
<tr>
<td>Computational ecology</td>
</tr>
<tr>
<td>Quantitative environmental science</td>
</tr>
<tr>
<td>Software design and development</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>16. SECURITY CLASSIFICATION OF:</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. REPORT</td>
</tr>
<tr>
<td>Unclassified</td>
</tr>
<tr>
<td>b. ABSTRACT</td>
</tr>
<tr>
<td>Unclassified</td>
</tr>
<tr>
<td>c. THIS PAGE</td>
</tr>
<tr>
<td>Unclassified</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>17. LIMITATION OF ABSTRACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>117</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>18. NUMBER OF PAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>117</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>19a. NAME OF RESPONSIBLE PERSON</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>19b. TELEPHONE NUMBER (include area code)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>