ELSEVIER

Environmental Modelling & Software 15 (2000) 169–177

Environmental Modelling & Software

www.elsevier.com/locate/envsoft

Design of multi-paradigm integrating modelling tools for ecological research

Ferdinando Villa *, Robert Costanza

Institute for Ecological Economics, University of Maryland, Center for Environmental Science, P.O. Box 38, Solomons, MD 20688, USA

Received 25 February 1999; accepted 21 July 1999

Abstract

Integrating modelling tools allow different modelling paradigms to coexist and cooperate in the same simulation model. The need for such tools in ecological modelling is due to the high level of complexity of ecological and environmental decision-making problems, their multiple scales of description, the diversity of the available approaches, and the size and heterogeneity of the available datasets. This article discusses problems and perspectives in developing integrating modelling tools and introduces the Simulation Network Interface (SNI), a software package for easy coordination of different existing simulation models. The interface allows the coordination of independent simulation models residing on different machines into higher-level, multi-paradigm, distributed simulation, with minimal recoding efforts of existing models. The interface can also be used to easily provide a remote interface to simulation or data retrieval services running on different architectures. As examples of its application, we describe three ongoing projects using the SNI: (1) the integration of Swarm, an agent-based simulation toolkit, with the Spatial Modelling Environment (SME), a process-based spatial simulation toolkit; (2) the straightforward implementation of a GIS-based spatial data repository for network-based data retrieval and manipulation; and (3) a network-based calibration service for complex simulation models.

Keywords: Multi-paradigm ecological modelling; Remote simulation control; Simulation interface design; Model coordination

Software availability

Name of software: Simulation Network Interface (SNI)
Related software: Swarn/SME interface classes (http://iee.umces.edu/~villa/swarmsme)
Online documentation: http://iee.umces.edu/~villa/sni
Developer: Ferdinando Villa
Contact information: Institute for Ecological Economics, University of Maryland, P.O. Box 38, Solomons, MD 20688, USA. Tel.: +1-410-326-7446; fax: +1-410-326-7354; email: villa@cbl.umces.edu
Year first available: 1998
Hardware required: Unix or Linux workstation for the server side; client library is portable to Intel and Macintosh PCs
Software required: C/C++ compiler. Java and Tel development kits are optional. Tcl and Expect libraries are required to compile the SNI server
Program language: Server: C++. Client library: C, Java. Easily portable to other common scripting languages (Tcl, Perl, Python)
Program size: 1.5 MB uncompressed tar archive. Library and executable sizes depend on platform
Availability: Licensed through the GNU General Public License (GPL) and available free in source form for all non-commercial purposes

1. Introduction

Environmental and ecological simulation modelling has reached a stage where a variety of approaches are available to study problems defined at different scales of time, space and complexity. Each of the dominant modelling approaches has stimulated the development of software tools that allow one to simulate specific systems by assembling generic "building blocks" modelled after

* Corresponding author.

1364-8152/00/$ - see front matter
PII: S1364-8152(99)00032-8

the main concepts of one approach. This is the case, for example, of individual-based modelling tools (DeAngelis and Gross, 1992; Minar et al., 1996) and spatially-explicit landscape modelling (Costanza et al., 1990; Maxwell and Costanza, 1997a,b).

The increasing complexity and multidisciplinarity of environmental research and management problems, the spatial and cultural delocalization of research groups, and the increasing recognition of the need for a multiplicity of scales to be considered at the same time, are reasons for the need to integrate different modelling approaches into higher-level simulation models. For example, predictive modelling of the evolution of land use under different management scenarios requires both a process-based landscape dynamics model and an individual-based model of the stakeholder community to coexist and interact.

Model integration is also a means of developing successful collaborative research projects. Since different models are typically developed by different research groups, the availability of easy-to-use integrating tools encourages collaborative development of ideas and approaches, facilitating at the same time the identification of common research endeavours using existing models.

The importance of integrative, collaborative tools in an increasingly complex modelling scenario has also less obvious justifications. In the case of ecological modelling, there is the distinct danger of tools preventing innovation. Ecological science needs new concepts to formalize and understand the complexity of nature. Modelling approaches can be thought of as providing metaphors which are relevant in this regard (Villa, 1992). It is thus very important that tools do not constrain the researcher's thinking space within a specific view of natural complexity, but rather allow free space for thought by endorsing knowledge models which allow flexibility and reorganization. Complex computer-based modelling tools can make the use of existing approaches easier, but they can also make the development of new approaches and ideas more difficult, because of the constraints imposed by the approach or the interface. Developing integrative tools is a way to add flexibility and potential for reorganization without losing the convenience and power of existing modelling tools.

In this paper we describe an approach to the integration of different independent models through a client–server approach which we have denominated Simulation Network Interface (SNI). The approach allows different simulation models, running on potentially different machines, to communicate through the network with a "master" coordinating model or interface, with minimal coding efforts. High-level, integrated simulation programs coordinating different concurrent simulations can be simply developed using the SNI Application Programming Interface (API). The system can be used, unmodified, to develop a variety of services, including the implementation of web-based simulation interfaces, data retrieval systems, and remote model calibration services. In the following, the SNI is described and real-life examples of the applications listed above are briefly illustrated.

2. The Simulation Network Interface

The rationale underlying the design of the SNI is the need of interacting and exchanging data between different simulation programs on heterogeneous, possibly remote machines through a simple set of calls. A key priority in its development was that "compliant" simulation models (i.e., models that can be coordinated remotely through the SNI) can be implemented in any language, with no need to link additional libraries, and that the recoding effort needed when adapting legacy models be small, or none at all. In fact, the complexities of the network interaction are entirely ignored by the programmer of the model, who simply concentrates on providing models, when necessary, with a command-line interactive interface which is used by the SNI server to control the simulation. Most models can be adapted to the SNI requirements with a few lines of code and do not require any additional libraries or programming tools. This allows immediate interoperability between old and new models without having to recode or significantly modify the old model. These design requirements are in contrast with the complexities of other current, general interoperability solutions like CORBA (OMG, URL) or Globus (Foster and Kesselman, 1997), whose adoption inevitably involves steep learning curves and requires substantial redesign of applications along with the installation of sophisticated and complex software infrastructure.

In the SNI approach, we distinguish "master" applications, which usually provide simulation scheduling and coordination, from "slave" simulation programs that are run by a remotely controlled program (the SNI server) through their command-line interface. Only master models need to use SNI client library calls (simple functions available in a variety of common programming languages) to control and coordinate remote simulations; "slave" models typically require no redesign other than the addition, when necessary, of a command-line interface. The latter feature is simple to implement, does not rely on any SNI-specific functions, and is a generally desirable and useful feature for most simulation models.

The Simulation Network Interface is implemented through two software components, the *SNI server* and the *SNI client library*. Both implement and use the *simulation network protocol* (SNP), a macro language transparent to the user, allowing communication and data transfer between independently run simulation programs.

The SNI server is a program residing on any host machine where simulation programs designed to be coordinated are run. "Master" simulation programs wishing to interact with a remote simulation will invoke SNI client library routines to start the SNI server on the remote machine. After access authentication, the SNI server will locate the required simulation program and run it. The operation of the slave simulation program is then controlled remotely by the master program through client calls to the server, which will in turn communicate the commands to the simulation program and return results, data and status information back to the master.

Fig. 1 illustrates an example scenario where two simulation programs and one GIS-based spatial database are scheduled and coordinated through the SNI by a master simulation driver. In the example, all programs reside on independent, network-connected host machines. Hosts 2, 3 and 4 run the SNI server, a program controlling different programs on each host. On host 2, the model is an agent-based simulation controlled by the master program on host 1. On host 3, the model is a grid-based dynamic landscape model, acting both as a slave (controlled by the model on host 1) and as a master (controlling the GIS on host 4). Host 4 runs a Geographical Information System (GIS) controlled by the SNI server through its command-line interface. The simulation program on host 3 also uses the SNI client library to connect to host 4 and retrieve lansdcape data from the GIS database. On host 1, a "master" program uses the SNI client library to schedule and coordinate the data exchange between simulation programs and the GIS, invoking the necessary actions from different programs according to the course of the coordinated simulation. The same scenario could run on one machine, running multiple copies of the SNI server, with no modification of the source code for all models. The master program on host 1 could run on a UNIX workstation as well as on a PC, or from a Java applet in a Web browser, allowing

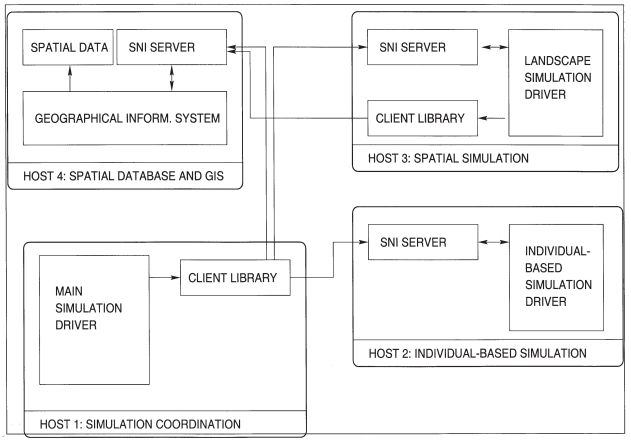

Fig. 1. A possible scenario with two simulation models and one GIS system cooperating through the SNI.

interoperability between different platforms and operating systems.

To be usable within a scenario like the one in Fig. 1, "slave" programs only need to comply with a few simple rules. Through the SNI client library, master programs instruct remote SNI servers to control other programs through a command-line interface of the same kind used for human interaction. Coordinated "slave" applications just need to be able to perform their basic operations (like stepping the simulation or outputting data to the terminal) as a response to a command issued by the user at the simulation program's prompt. The SNI server transparently replaces an interactive user and controls the simulation through the command-line interface. Error, warning, and informational messages are output to the terminal according to a simple, flexible syntax. The SNI server filters such messages from the program's output, storing them for access and notification in the master program. See the SNI documentation (Villa, 1998 and online) for the operational details of model integration.

Master programs (those controlling the scheduling of the slave programs) use SNI client library functions to control and access the dataspace of the coordinated simulations. As explained in more detail later, the SNI Application Programming Interface is simple to implement in most languages, and is available at the moment in C and Java.

A macro language (the SNI protocol: see Villa, 1998) is used internally in the SNI architecture to communicate with the server, transmit commands to slave programs and retrieve data and status information. Client calls invoked by a master program use the SNI protocol to invoke actions from the SNI server. As an answer, the server runs programs, interpreting and storing their status and their output when commands are issued to them. As mentioned, error and warning messages are filtered by the server and organized for subsequent notification and retrieval in federated programs through SNI protocol calls.

2.1. The SNI architecture

Fig. 2 illustrates schematically the SNI components. The different components and operations of the SNI architecture are described in more detail in the following.

2.1.1. Host authentication and server startup

Client library routines are used to connect to remote hosts where the SNI server has been installed. Upon request, the UNIX inetd service daemon starts the server which performs a set of authentication operations. The configuration of the SNI server on each host allows host-based authentication (with lists of allowed and denied hosts and domains) and password-protected user-based

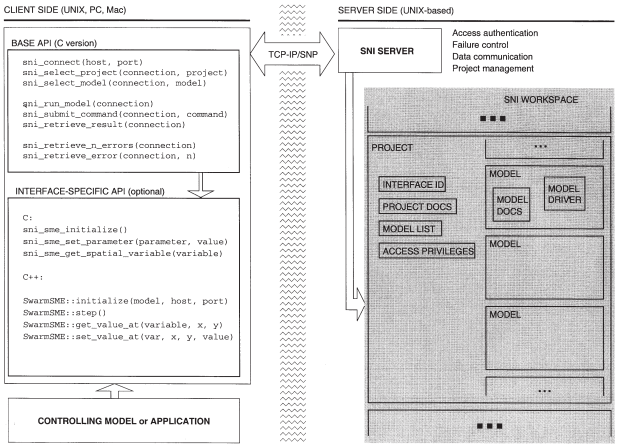

Fig. 2. A detailed scheme of the SNI operations. On the client side, some functions from the base C API and two hypothetical tool-specific APIs are shown. On the server side, a diagram of the SNI workspace illustrates the arrangement of "slave" simulations in projects, including documentation and interface specifications.

authentication with different access privileges. Different access privileges to specific projects, according to which host and user are connecting, are supported, in order to allow widespread access to "demo" models as well as selective access to resource-critical simulation services located on the same host.

2.1.2. Workspace management

The SNI server manages a "workspace" (mapped on a directory tree in the filesystem of the host machine) where SNI-enabled "slave" models are organized into higher-level collections called "projects", as illustrated in Fig. 2. Projects group together models of similar scope and application, using the same interface (see below) and sharing some high-level documentation. Through SNI client library calls, master programs can obtain lists of the projects available on the host (based on their access privileges) and lists of the models within a particular project. Documentation can be retrieved in textual form for both the project and each particular model within it. We are successfully using the eXtensible Markup Language (XML: Harold, 1998) to structure documentation for projects and models in the SNI workspace.

2.1.3. Interface selection

Simulation programs created with the same tool (e.g., SME: Maxwell and Costanza, 1997b; Swarm: Langton et al., URL) usually employ similar startup, initialization, and interaction protocols. The SNI server needs to know how to start, initialize and run each particular model in order to make its remote operation easy and consistent. The set of informations needed to perform these tasks (such as the format of the command line used to start the program, the user prompt employed by the interface, etc.) are grouped into an interface specification and stored as a named unit in the server configuration file. The name of the interface is stored in a file in each project's workspace, so that the proper interface can be loaded in the server automatically upon project selection. This way, the implementor of the simulation program is free to employ custom rules, commands, and program invocation details, and will only need to specify these details once for each simulation toolkit supported. Interface names can be read through SNI calls in order to provide interface-specific remote services to SNI-controlled programs: for example, generic graphical interface tools (such as the Collaborative Modeling Environment, CME: Villa, 1997b; Voinov et al., 1999) can provide a Graphical User Interface (GUI) for SNI-controlled models, selecting the right interface for the tool upon project selection. Interface specifications can be made flexible and configurable by using "server variables", named variables whose values are set by master programs through SNI client calls, and are macro-substituted in interface specifications at the time of use.

2.1.4. Remote control of slave programs

After a simulation project, model and interface are selected among those known to the server, the model driver is started and its standard input and output streams are connected to a pseudo-terminal controlled by the SNI server. Client routines are then used to submit commands and retrieve the program's answers through the SNI protocol. Multiple connections to the same host are served by independent instances of the SNI server, each of which can employ different interfaces to access different projects and models. The Application Programming Interface (see below) provides generic functions to submit commands to the model driver and retrieve its answer in a variety of formats, as well as checking its error output. Exceptional behaviours such as model driver crashes or timeouts are intercepted and notified to the master program.

2.1.5. Data transfer

Data transfer within the SNI architecture involves two processes: communication of data between simulation program and SNI server, and communication between SNI server and SNI client calls. Different methods, implementing different levels of efficiency and complexity, are available within the SNI framework to transmit data bi-directionally between the server and the master simulation program. In the simplest case, data in ASCII form are output on the program's command line and read by the server. This is usually efficient enough to transfer even relatively large amounts of data from the simulation to the server. Different encoding methods can be used to optimize the efficiency of data transfer when the amount of data is very large. Simple run-length encoding of the command-line output is often an adequate way of obtaining efficient transfer of large data items and involves a minimal programming overhead.

The SNI is distributed with functions implementing several encoding algorithms, to be used within the slave simulation program if necessary. The server can recognize data encoded with any such methods and translate them appropriately.

The transfer of data across the network between the SNI server and the client routines in the master simulation can use different encoding schemes according to the nature and amount of data to transfer. Issues of data transfer efficiency have different importance according to the size of the datasets and the efficiency of the network connection. The selection of the best encoding algorithm is done in the SNI server in order to optimize the trade-off between the time required for encoding and the time required for transmission. The encoding algorithms are used transparently so that client libraries can send and receive data in raw form. Different client functions are available to retrieve and send data in the form of single values, lists, and *n*-dimensional arrays of integer and floating point values.

2.1.6. Application Programming Interface (API)

Although the core set of client API functions provided with the SNI toolkit can drive any SNI-compliant program and exchange data with it, the most productive approach to the use of SNI-coordinated applications is usually to write a set of specific client routines for each particular simulation tool, encapsulating tool-specific concepts so that the additional set of routines behaves exactly like a local simulation model. This is very easy to do using only the core SNI API, which implements generic command submission, retrieval of results, and transfer of data. As an example, specific API functions can be written for a simulation tool to step the simulation, retrieve the values of variables, or specify model parameters, as shown in Fig. 2. The tool-dependent API implementation is usually very small and can be written in any language able to access one of the core API implementations. The ANSI C implementation of the core API can be used in a variety of language environments such as C, C++, Objective-C, Tcl, Perl, Python, Eiffel. An object-oriented, native Java implementation is also included with the tool distribution, suitable for use with Java applications and within Java-enabled Internet browsers. Detailed examples of tool-specific APIs are described in the SNI documentation (Villa, 1998).

The reliance on a command-line approach for slave simulation programs makes it easy to exploit the potential for intermediate processing of data and advanced simulation control coming from the use of widely available integrated language interpreters, such as Tcl (Ousterhout, 1994), Perl (Wall et al., 1996), or Python (Lutz, 1996), as a toolkit to develop command-line interfaces for the simulation programs. By using these simple, yet full-featured languages, SNI-based applications can ask the "slave" simulations to perform very complex pre- and post-processing tasks, by submitting the correspondent language code to the program through the SNI server.

2.2. Applications of the SNI

By virtue of the "naivety" of the approach, which only imposes minimal constraints on cooperating programs, the SNI architecture is suitable to a variety of useful applications. The following is a discussion of its possible uses in the current implementation.

2.2.1. Remote simulation services

The SNI is a natural choice to develop remote interfaces to simulation programs. By using only the SNI client functions, client programs running on simple personal computers can drive and display data from simulations running on remote, more powerful workstations. The computation service is kept conveniently separate from the interface. The Java implementation of the SNI client libraries is being used at the University of Maryland for the development of simulation interfaces running as applets within Internet browsers for educational purposes (Maxwell et al., URL).

2.2.2. Development of coordinated simulation models

Two approaches to the integration and coordination of different simulation models are possible. The integration of different models can be performed at the executable level or at the language level. In the latter case, each separate model is recoded into a common language framework. This enforces modularity, scalability, and integration among model components, since each model needs to be expressed within the same semantics. The advantages of this approach in terms of consistency and potential for further development of models are obvious. Nevertheless, keeping such a framework general enough to be suitable to multi-paradigm modelling is not trivial; the implementation details are likely to be so overwhelming to require a complete rewrite of most existing models. Another problem comes from the lack of realism implicit in the assumption that, even in the case that an entirely general semantics could be defined, a sufficient number of researchers would be willing to adopt it, in view of their different culture, needs, and ways to conceptualize the problems. What is more important, proposed standards for such languages are just being developed and implementations are not yet available (Fritzson and Engelson, URL). Other available or proposed languages are restricted to more specific modelling paradigms (Maxwell and Costanza, 1997a).

The approach implemented in the SNI works at the executable level, using existing models implemented as separate, "federated" executable programs. This has the advantage of being easy to implement and better suited to the reuse of existing running code. Programs do not

have to adhere to any particular conceptual structure to be usable within the SNI. One of the programs acts as a "master" and supplies the scheduling and scale translation required to effectively coordinate a multi-paradigm simulation. A variety of schemes can be implemented. In the simplest, a simulation program controls another, retrieving and modifying its dataspace and stepping the remote simulation as an additional action in its own schedule of events. More complicated scenarios can be implemented by writing specific "driver" programs which control a federation of models and organize the transfer of data and, when necessary, their cross-scale translation. In all cases, such programs are simple to write with the SNI API, and open new collaborative horizons to existing simulation models where the resources for a language-level integration are missing.

2.2.3. Implementation of data servers

Data can be thought of as the simplest model of a phenomenon, embodying the lowest amount of added knowledge, but still dependent on assumptions, scale choices, and an abstract model of reality which inspires their selection, measurement, and collection. In view of this fundamental continuity it seems appropriate to view a data server as an implementation of the simplest modelling paradigm, which in general needs explicit coordination with simulation tools to be useful in the context of a multi-paradigm simulation model. There are many ways to implement remote data servers which are particularly suited to ecological simulation, like object-oriented database servers or network-based data repositories. The SNI interface can be used to provide remote control functionalities to programs like Geographical Information Systems (GIS), so that cooperating programs can access GIS services through the same interface used for accessing the simulation services. An example of this is illustrated in the next section.

3. Examples

3.1. Integration of Swarm with the Spatial Modelling Environment

Two of the major paradigms now prevailing in ecological simulation are individual-based simulation modelling (DeAngelis and Gross, 1992) and large-scale process-based spatial modelling (Costanza et al., 1990; Voinov et al., 1999). In both cases a number of software tools are available to define, develop and run simulation models. A software system supporting the individual-based abstraction is Swarm (Minar et al., 1996; Langton et al., URL), developed by Chris Langton and colleagues at the Santa Fe Institute. In the Swarm framework, the modeller uses the conceptual structures of the object-oriented Objective-C language to define the behaviour of the individual agents involved in the simulation. Agents can be of different kinds, e.g., observer agents are used to "probe" other agents, and space agents implement a virtual landscape for other agents to move in. A Swarm simulation program creates a set of collections (swarms) of agents and schedules their actions at every step of the simulation.

The Spatial Modelling Environment (SME: Maxwell and Costanza, 1997b; Maxwell et al., URL) translates process-based dynamic models expressed in common formats like STELLA (Costanza et al., 1998) into an intermediate Modular Modelling Language (MML: Maxwell and Costanza, 1997a) representation. This representation is then translated into C++ and linked with the spatially-explicit objects in the SME library to create a landscape model with multiple resolution views of the same rasterized physical space, with a locally parameterized instance of the original "unit" model running in every raster cell.

While each of the systems is useful on its own, the synergistic power coming from their federation is obvious. Individual Swarm agents can observe and influence the evolution of the dynamic SME landscape, each with a potentially different perception and influence on the landscape. For example, Swarm agents could represent individual deers, families, stakeholders and managers, living and operating in different ways and with different perception of the changes in the landscape, whose process-based dynamics could be simulated by an SME model.

As a first step in integrating the two approaches, a client API for the SME driver has been written in C, using the core SNI API, to provide a set of functions which can drive a remote SME simulation implementing the SME concepts directly. Since SME simulation programs already incorporate a command-line interface based on the Tcl language (as described in Villa, 1997c), no changes are needed to make them compliant with the SNI requirements. The SME interface was specified in the server configuration file listing details of how the programs are to be invoked, the prompt string they output, and so on.

The Swarm module acts as the master simulation and provides the scheduling for the SME landscape model. To allow this, a Swarm class to represent a SME-driven dynamic space was derived by the simple two-dimensional space classes already implemented in Swarm, using Swarm's Objective-C API and integrating the SME-specific C functions. This new space represents a dynamic, multiple-resolution landscape with variables whose values change in time, and is initialized with the host, project, and model names corresponding to the SNI-controlled SME model. Subclasses of the dynamic space class are derived for different SME models. Upon startup, the (possibly remote) SME model is started and two-dimensional Swarm grids are created to represent

each variable in the spatial simulation. Swarm agents can access the value of each variable at every site of the landscape and ask the SME simulation to provide statistics about particular aspects of the landscape (like the values of spatially distributed variables). Data transfer between SME space and Swarm agents can be scheduled automatically after every step of the spatial simulation or performed on demand. Two-way transfer of data allows SME state variables and parameters to be modified by Swarm agents. Other functions available through the Swarm–SME space class are used to control the flow of the SME simulation, performing new steps according to the scheduling determined on the Swarm side.

Since the SME runs a command-line interpreter implemented as an extension to the Tcl language (as explained in Villa, 1997c), new functions returning aggregated or post-processed data about the landscape can be implemented in Tcl directly without the need to modify the SME model. The SME-specific API has functionalities to submit the definition of new function to the SME Tcl interpreter from the client side.

The Swarm–SME integration class are available through the Swarm code repository or by contracting the first author.

3.2. A remote spatial data server

As mentioned above, the availability of a remote server for processing and retrieving spatial data is useful for both providing remote interfaces to the usually huge and complex spatial data processing and storage systems, and to provide simulation programs with a way to retrieve and store configuration data without having to physically move the archives from where they are stored, and without having to deal with translating GIS file formats. Such a mechanism is currently possible on the same host where the GIS system is located, by linking in proprietary APIs to systems such as Arc-INFO (ESRI, URL) or GRASS (OGF, 1994).

The SNI offers a clean and particularly easy solution to this need by allowing remote access to the GIS through a specially written, command-line based program following the rules for SNI compliance outlined above. We are testing a module for GRASS that allows retrieving, modifying and processing data stored in a GRASS database, to be used as an SNI model on a machine where GRASS is available. The module works through a command-line interpreter which can be run by either a human operator or the SNI server. Compared to the standard GRASS command-line interpreter, the SNI GRASS module just adds commands to output the spatial data in a convenient encoding for efficient transmission to the SNI server. In this simple form, simulation programs can connect to a site where the GRASS interface has been installed as an SNI project, and use the system transparently, through SNI client calls embedded in a GRASS interface library, to retrieve or store data and to invoke processing services. The system can be used in the same way by simulation programs or by remote graphical interfaces to the GRASS system, either stand-alone or web-based.

The GRASS SNI driver is under development and will be made available through the GRASS contributed code repository or by contacting the first author.

3.3. A remote model calibration service

An aspect that is central to the development of useful models is parameter optimization. Many complex optimization techniques are available but their implementation requires very specialized skills and powerful computational resources. As a result, many sophisticated models used in ecological research are still calibrated by trial and error, with a high chance of losing most of the interesting range of dynamic behaviours that a complex model can show across the parameter space. Standardization of model evaluation techniques and availability of centralized optimization services are much needed conditions for a systematic adoption of formal calibration, which is necessary to avoid under-utilization of the power and sophistication of the current generation of environmental models (Villa and Costanza, 2000).

The SNI architecture is being used in the development of network-based calibration services, which can be accessed by any simulation program driven by the SNI server. In the calibration service we are implementing, a server machine runs remote simulation programs according to the specifications sent by the user of the service and implementor of the program. The latter has to specify the following by filling in a web-based form:

- The network location or values of the calibration reference data.
- The calibration criteria, expressed as an on-line definition of an objective function based on our Model Performance Index (MPI: Villa, 1997a).
- The commands to be sent to the SNI server in order to start the simulation model, set parameter values, run for the necessary amount of time, and retrieve the output data necessary for calibration.
- The names and ranges of the parameters the user wants calibrated.

The calibration program is then invoked by submitting the web-based form. The user is presented with a control page through which a variety of calibration algorithms can be selected and configured. These include genetic algorithms (GA) for global optimization and hill-climbing local search techniques to improve on the GA estimates. The calibration results consist of sets statistics and parameter sets, which are presented in a separate frame appearing in a web browser. Each final set of para-

meters is given an ID and stored on the server side, so that the user can select a particular set as the starting point of a different calibration run.

The generality allowed by the adoption of the SNI interface and a standardized definition of the objective function allows access to calibration services without having to write optimization programs on purpose. All that is needed is to make one's program compliant with the SNI by providing it with a simple command-line interactive interface. A prototype of this network-based simulation service is being developed by the first author, to whom inquiries on future availability should be directed.

4. Conclusions and perspectives

In its simplicity, the SNI interface provides an effective answer to the need to integrate different modelling tools allowing potentially different modelling paradigms to interoperate. As discussed, this approach has multiple benefits for environmental research, from the added potential in developing ideas involving collaborative use of independent research products to the side benefit of allowing old approaches to be reconsidered and re-evaluated in a fresh, collaborative context.

We have argued that a realistic perspective on the use of integrating modelling tools calls for an easy-to-use approach which does not require modellers to: (1) recode existing models; (2) learn complex software interfaces; and (3) be forced to adopt specific modelling approaches and description languages. While we believe that all of these issues, and in particular the definition of an interoperable semantics to address multi-paradigm problems, are urgent and important research topics, we do not see in this approach a ready solution to the problem of creating multi-paradigm models. With these needs in mind, we have developed the SNI architecture which offers a maximum of flexibility with a minimum of development cost and requires only basic programming and system managing skills to implement. We have shown how it can be used to solve problems and develop approaches that are currently in demand.

The concept of an integrating environment to drive and connect running simulations can be expanded to higher levels of user abstraction to implement graphical integrating tools using familiar interface concepts. The SNI environment for the Collaborative Modelling Environment (CME: Villa, 1997b; Voinov et al., 1999) is being developed as a generic graphical front-end to any SNI workspace, allowing the creation of custom graphical interfaces for each SNI-compliant modelling tool, and the storage of their description on web-accessible repositories. This will allow us to expand the potential even further: as soon as new interfaces for simulation tools are developed, their CME-based interfaces can be made public and retrieved automatically by the software. We see these research efforts as steps towards a higher level of integrated, multi-paradigm ecological modelling.

Acknowledgements

The Swarm/SME integration classes have been developed with support by the U.S. Army grant DACA88-98-M-0232. The Simulation Network Interface has been developed in the context of the NSF 784AT-31057A PACI Alliance project, Subaward #784. Thomas Maxwell and an anonymous referee provided helpful comments and suggestions. Thomas Maxwell also wrote the Java SNI client API.

References

Anon., 1994. Grass 4.1 User's Manual. Open GRASS Foundation (OGF).

Costanza, R., Sklar, F.H., White, M.L., 1990. Modelling coastal landscape dynamics. BioScience 40, 91–107.

Costanza, R., Duplisea, D., Kautsky, U., 1998. Ecological modelling and economic systems with STELLA — introduction. Ecological Modelling 110, 1–4.

DeAngelis, D.L., Gross, L.J. (Eds.), 1992. Individual-based Models and Approaches in Ecology: Populations, Communities, and Ecosystems. Chapman & Hall, New York.

ESRI (URL) ArcInfo. http://www.esri.com/software/arcinfo/index.html.

Foster, I., Kesselman, C., 1997. Globus: a metacomputing infrastructure toolkit. Supercomputer Applications 11, 115–128.

Fritzson, P., Engelson, V. (URL) Modelica — a unified object-oriented language for system modelling and simulation. http://www.ida.liu.se/~vaden/paper/modelica/index.html.

Harold, E.R., 1998. XML: Extensible Mark-up Language. IDG Books Worldwide, Foster City, CA, USA.

Langton, C., Burkhart, R., Daniels, M., Jojic, V., Lancaster A. (URL). The Swarm Simulation System. http://www.santafe.edu/projects/swarm

Lutz, M., 1996. Programming Python. O'Reilly and Associates, Sebastopol, CA, USA.

Maxwell, T., Costanza, R., 1997a. A language for modular spatio-temporal simulation. Ecological Modelling 103, 105–113.

Maxwell, T., Costanza, R., 1997b. An open geographic modelling environment. Simulation Journal 68, 265–267.

Maxwell, T., Villa, F., Costanza, R. (URL). Spatial Modeling Environment. http://iee.umces.edu/SME3

Minar, N., Burkhart, R., Langton, C., Askenazi, M., 1996. The Swarm Simulations System: a toolkit for building multi-agent simulations. Santa Fe Institute Working Paper 96-06-042. Internet: http://www.santafe.edu/projects/swarm/swarmdoc/swarmdoc.html.

OMG (URL). Object Management Group home page. http://www.omg.org.

Ousterhout, J.K., 1994. Tcl and the Tk Toolkit. Addison-Wesley, New York, NY, USA.

Villa, F., 1992. New computer architectures as tools for ecological thought. Trends in Ecology and Evolution 7, 179–183.

Villa, F., 1997a. Usage of the Model Performance Evaluation software. Internal report, Institute of Ecological Economics, University of Maryland. Internet: http://iee.umces.edu/~villa/svp.

Villa, F., 1997b. Usage of the Collaborative Modelling Environment.

Internal report, Institute of Ecological Economics, University of Maryland. Internet: http://iee.umces.edu/~villa/cme.

Villa, F., 1997c. Guide to the Spatial Modelling Environment TCL-based command line interface. Internal report, Institute of Ecological Economics, University of Maryland. Internet: http://iee.umces.edu/~villa/sme_tcl.

Villa, F., 1998. Stimulation Network Interface: user guide and reference manual. Internal report, Institute of Ecological Economics, University of Maryland. Internet: http://iee.umces.edu/~villa/sni.

Villa, F., Costanza, R., 2000. Performance and goodness of fit of complex simulation models: a multi-criteria approach to model calibration using the Model Performance Index (MPI) framework. To appear in: Costanza, R., Voinov, A. (Eds.), Spatial Ecosystem Modeling. Springer-Verlag, in press.

Voinov, A.A., Costanza, R., Wainger, L.A., Boumans, R.M.J., Villa, F., Maxwell, T., Voinov, H., 1999. Integrated ecological economic modelling of watersheds. Journal of Environmental Modelling and Software 14, 473–491.

Wall, L., Christiansen, T., Schwartz, R.L., Potter, S., 1996. Programming Perl, 2nd ed. O'Reilly and Associates.