Objects for simulation: Smalltalk and Ecology

Baveco, J.M.; Smeulders, A.W.M.

DOI
10.1177/003754979406200106

Publication date
1994

Document Version
Final published version

Published in
Simulation

TECHNICAL ARTICLE

SIMULATION 62:1, 42-57

ISSN 0037-5497/93 $3.00 + .10
Printed in the United States of America

Objects for Simulation: Smalltalk and Ecology*

J.M. Baveco
Dept. of Pure and Applied Ecology
working group on Biological Information Techniques
University of Amsterdam
Kruislaan 320; 1098 SM Amsterdam,
The Netherlands

A.M.W. Smeulders
Dept. of Computer Systems
working group on Biological Information Techniques
University of Amsterdam
Kruislaan 320; 1098 SM Amsterdam,
The Netherlands

The object-oriented programming environment Smalltalk is used to implement a tool for modeling and simulation of ecological systems. This PC-based tool makes it possible, and easy, to represent individuals explicitly in the simulation, and to integrate the individual-based approach with a traditional population/concentration-based approach. Object-oriented programming is used to allow for an efficient development of models. The properties of Smalltalk are exploited to make testing and investigation of the models occur interactively, supported by a user-friendly interface. In the paper, modeling and simulation concepts and elements of object-oriented programming as they relate to an individual-based approach, are introduced. The simulation extension in question (EcoTalk) is described. Two applications that make use of EcoTalk are presented, both related to population-dynamics. The discussion centers on practical implications of the approach, e.g. the system characteristics that can be expressed in EcoTalk. An indication of the performance of an application is given. It is concluded that, at present, EcoTalk can be used for medium-sized applications.

keywords: discrete-event, ecology, GUI, individual-based, object-oriented programming, performance, population-dynamics, Smalltalk.

Introduction

In ecology, modeled systems are predominantly represented at population or concentration level. Models usually are sets of difference or differential equations. Reflecting the complexity of nature, in most cases they need to be solved numerically. In an ecological context this modeling approach has conceptual shortcomings. It is of limited use in dealing with systems characterized by abrupt events (discontinuities) and by interacting processes on multiple spatial and temporal scales. Furthermore, in population models the description of population behavior is usually derived from the properties of an average individual, in an average environment. System organization, emerging from the interaction between unique individuals and their local environment, is being ignored.

As an alternative, a combination of individual-based modeling and discrete-event simulation may overcome some of these limitations (see Figure 1.). A challenge to the approach of simulating individuals is that their numbers are variable and potentially high, and that their behavior is complex. Developments in computer hardware have brought this calculation-intensive approach within reach of the common computer user. New software technology, i.e. object-oriented programming, makes it feasible to handle the complexity of the associated programs. However, little software is available, tailored to the needs of ecologists (for some examples see Hogeweg & Hesper, 1990). In the context of a research project on variable-structure simulation, in

*Supported by a grant from Delft Hydraulics.

this paper a framework for individual-based simulation is described, called EcoTalk. It is intended to be an environment in which individual-based models can be developed and simulated rapidly. Integrated in the object-oriented programming system Smalltalk, it provides objects as the building-blocks for ecological models. EcoTalk is thus based on the following three concepts.

Individual-Based Modeling

A system is described in terms of local individuals, interacting with one another and their environment. Actions are not necessarily synchronous and different types of individuals may have their own characteristic timescale on which they operate. Although the rules for behavior are shared by a group of individuals (e.g. those belonging to the same species), the behavior is not identical for all, due to the interplay between internal properties and local circumstances. The ecologist thus can take into account individual and local variability, and study its relation to macroscopic (population-, community-level) system behavior (Huston et al., 1988). Individuals can be basic components of population, community and ecosystem models. When multiple steps in a food chain are involved, it is usually not feasible to represent all components at the individual level (for instance, fish, zooplankton and algae). For practical purposes, such a model should include individual-based representations of the entities at the highest trophic level, acting on large spatial and temporal scales. The (numerous) entities, operating on smaller scales, can be aggregated in population or sub-population units, e.g. super-individuals: model individuals that represent a large number of identical individuals. These units should be much more flexible than they are in traditional population-based models, i.e. they should allow splitting, lumping, etc.

Discrete-event Simulation

The asynchronous events and multiple timescales, characterizing individual-based systems, can be handled in discrete-event simulation. In discrete-event simulation, the simulation clock moves from event to event, with intervals of undetermined length in between. Periods with no activity in the system are skipped, while periods with intense activity imply a high density of events and consequently a slow progressing of simulation time. A discrete-event timing regime is compatible with the simulation of continuous processes. Continuous processes are usually simulated as synchronized state-changes occurring after a certain timestep. They can be represented by events, repetitively scheduled on a fixed timestep base.

Object-Oriented Programming

In object-oriented programming (OOP), objects are program modules defined as a combination of data and procedures. Data encapsulated in an object can only be changed by the procedures of this object. Objects communicate by sending messages (procedure-calls) to one another. Objects are abstract data types, that is, they can be used as building-blocks for new objects. Objects are organized in a hierarchy. This organization, based on inheritance and polymorphism, is characteristic of OOP. Each object inherits the properties of its parent object: its data-structure and procedures. However, an object may also overrule the definition of an inherited procedure (polymorphism).

Object-oriented programming's first aim is to master the complexity of large software systems and promote the reuse of software components. In simulation, the benefits of inheritance have long been recognized. Already in SIMULA, new models could be constructed efficiently, by defining them as subclasses of existing model classes. Object-oriented programming has been applied successfully to deal with the complexity of ecological systems (Larkin et al., 1988; Saarenma et al., 1988).

In simulations involving a large number of components, encapsulation of data inside objects is a relevant feature. In the context of the research presented here, additional arguments in favour of object-orientation include (1) the concept of an individual belonging to a species, nicely fits the definition of an object, adhering to the rules of its class but otherwise acting on its own local data, (2) the hierarchical organization of objects lends itself well to a representation of taxonomical relations between component species, (3) in an individual-based approach, the analogy between computation by message-passing and communication by signals can be exploited.

A one-to-one relationship between objects and realworld physical entities results from the object-oriented approach. The observation and animation of system behavior and the task of collecting data become easier, as the state of an object persists during the simulation as a coherent set of properties.

In this paper, EcoTalk will be considered from a practical point of view, evaluating its performance and expressiveness. From the few widely-available object-oriented tools for the PC, the Smalltalk environment was selected for implementation of EcoTalk, due to its extreme flexibility. First, elements of Smalltalk are discussed. Then, the functionality of EcoTalk is described. Next, some applications are shortly considered. The concluding section is devoted to the feasibility of the object-oriented approach.

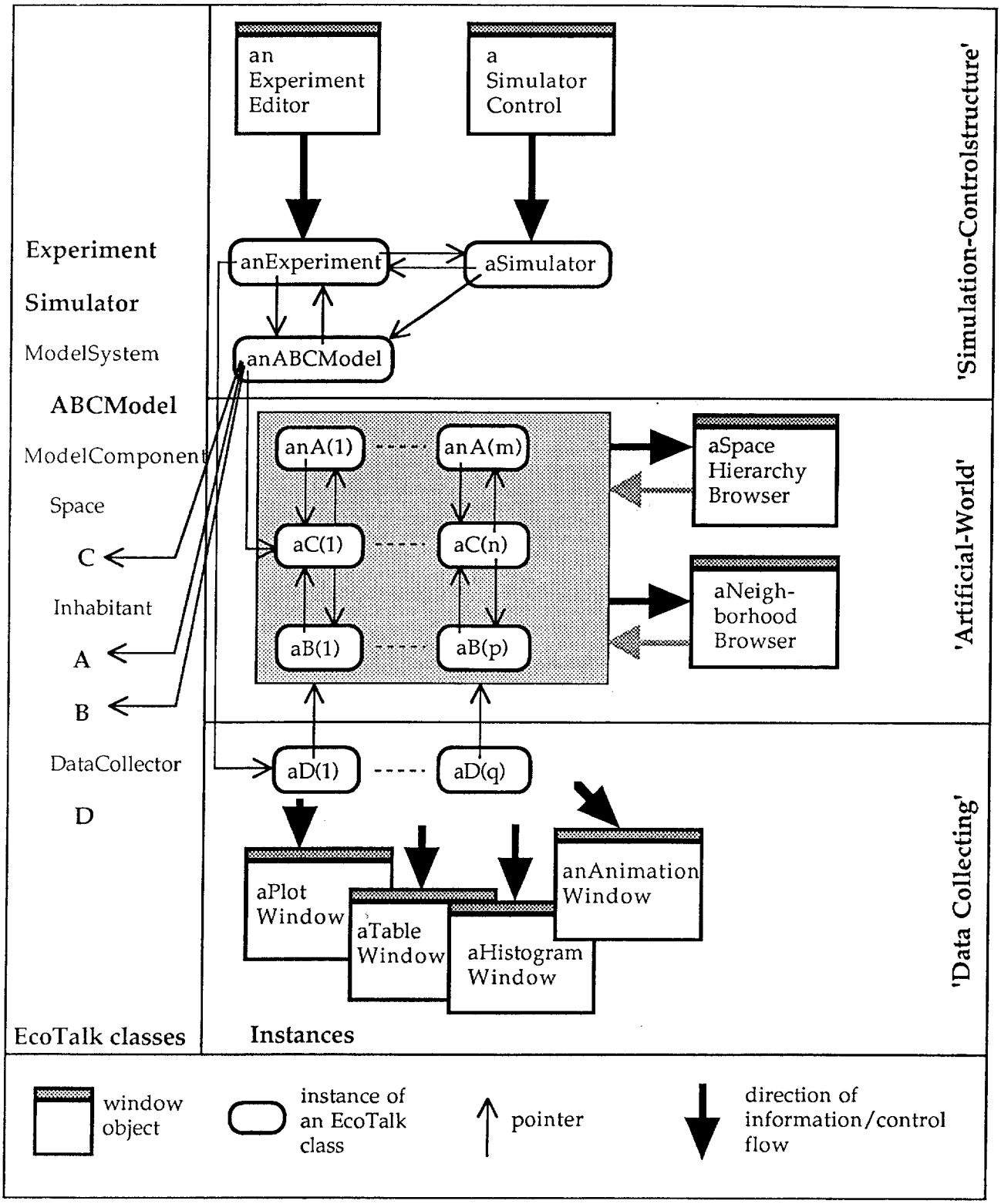


Figure 1. An EcoTalk simulation in action. Instances of several classes (left, bold) cooperate to maintain a simulation control structure (top level), to represent the real-world system (intermediate level), and to extract information (bottom level). The system represented here consists of m individuals of species A, p individuals of species B, in an environment made up of n elements of type C. A number q of data collectors of type D collect information. The main pointer structure is depicted, as well as the main flow of information/control between the simulation and the user interface.

originally represented object behavior by simulating parallel processes, synchronized by semaphores. Here, the process-interaction approach is left behind and replaced by single event-scheduling, as the latter involves less overhead and appears more transparent. Probability density distributions, needed in the generation of stochastic parameters, are added according to the description found in Goldberg & Robson (1983). An alternative to the standard (linear congruency) random number generator is also added (Kirkpatrick and Stoll, 1981).

Overview

The classes figuring in an EcoTalk simulation are divided into three categories (the levels in figure 1). At a basic level, they constitute a commonplace discrete-event simulation framework ('Simulation Control Structure'). A second level unites the classes with a counterpart in the real world ('Artificial World'). Finally, some classes serve to extract information from the simulations ('Data Collecting'). Each level also includes its user interface elements.

Figure 2 shows how these classes fit into the inheritance structure of the Smalltalk class hierarchy. Each particular model is shaped by a combination of one or more classes from the ModelComponent branch and one from the ModelSystem branch. All basic EcoTalk classes are abstract classes, that is, no instances of these classes are used in a user-defined model. In the following, the EcoTalk (kernel) classes are described in more detail. Names starting with a capital refer to instances of a class or to the class itself.

Simulation control structure

A simulation is considered to be an experiment performed on a model system. An experiment makes a relationship among three elements: a model, an experimental frame, and the generated data. A model consists of one or more components. The term experimental frame is used to denote model specific information on (1) observational variables, (2) input schedules, (3) initial setting, (4) terminating conditions and (5) data processing (Ören & Zeigler, 1979).

A simulation is an instance of class Experiment. Each model system is an instance of a subclass of ModelSystem. Each component is an instance of a subclass of ModelComponent. ModelComponents are atomic models (Zeigler, 1990), i.e. they cannot be divided any further in more elementary components. Some ModelComponent classes include information on the coupling relations between instances, for instance, ModelComponents representing spatial units are connected to one another (neighbors) according to fixed rules. Details on the implementation of these classes are given in figure 3.

In ModelComponent common functionality is contained by all simulation components, i.e. features making it possible to recognize and count individual active instances. The ability to schedule events in future simulation time is defined in subclass Actor. ModelSystem contains the properties shared by all simulation models.

ModelSystems refer to the involved component classes and the experiment they are subjected to. In all simulations a structured ecological environment is

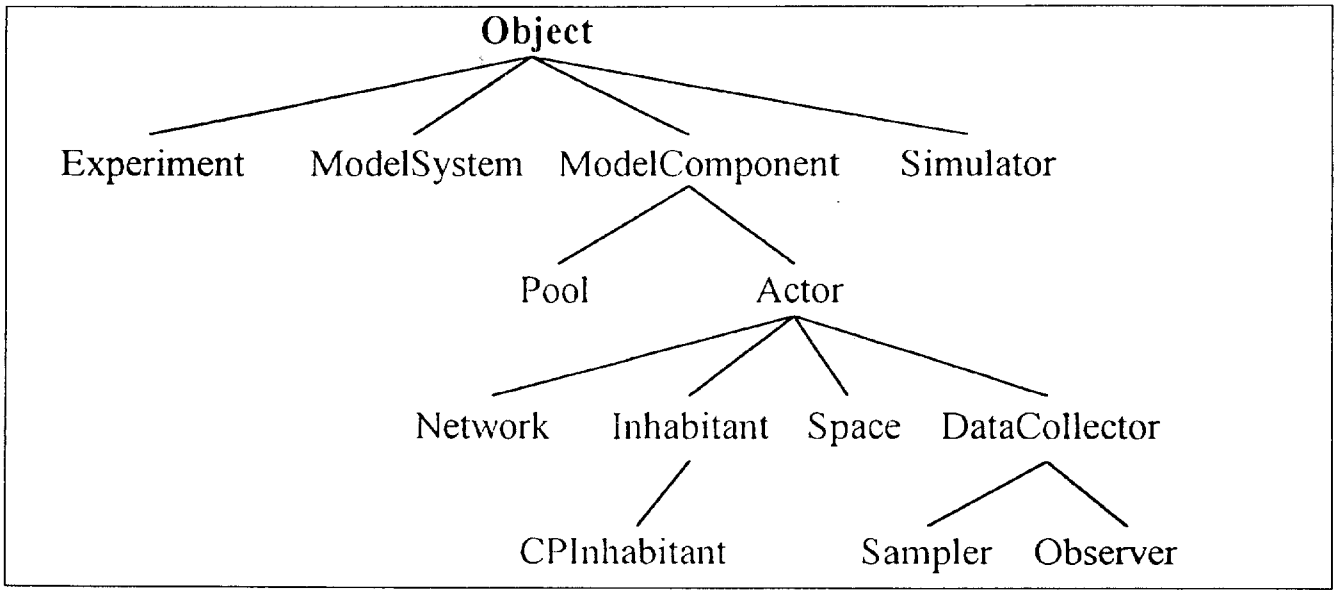


Figure 2. An overview of the hierarchical relations between the kernel EcoTalk classes. Object is the root class of the entire Smalltalk class hierarchy.

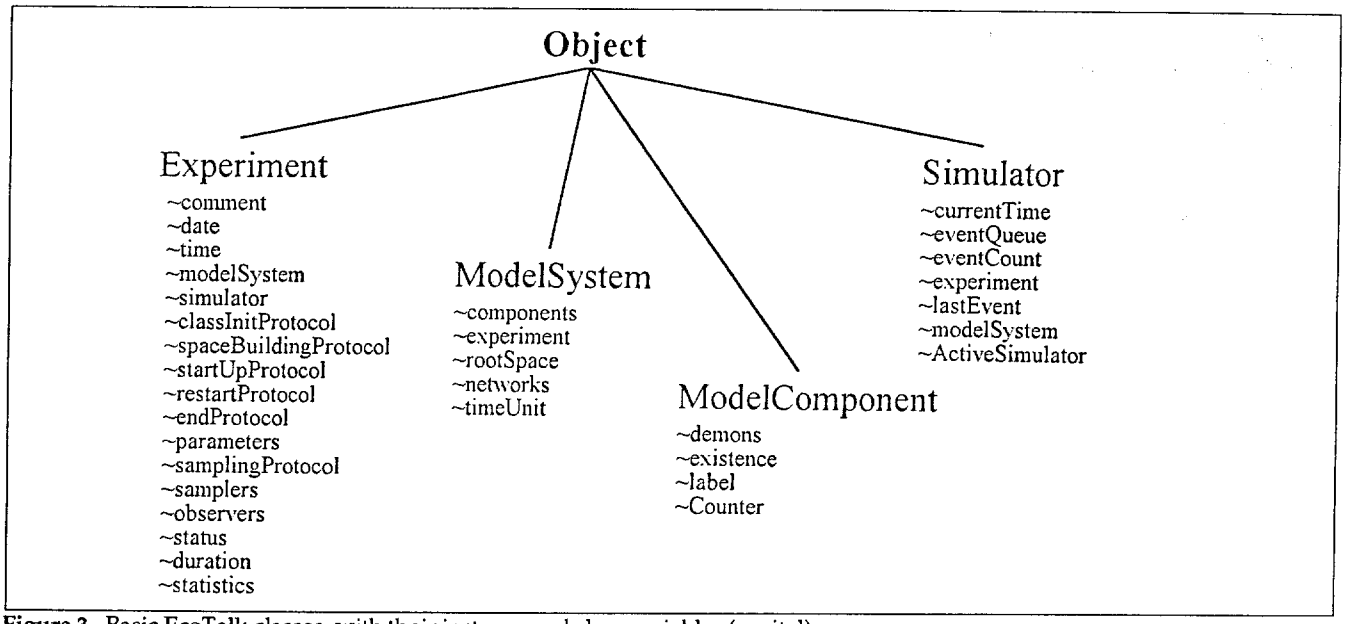


Figure 3. Basic EcoTalk classes, with their instance and class variables (capital).

assumed. This spatial structure can be reached via the ModelSystem. All model-specific methods are added in subclasses, e.g. defining parameter values, initial conditions, and the kind of data to be gathered. These methods define a default experimental frame for each specific model.

Class Experiment is used to define, run and end a simulation. An Experiment is started by connecting it to a ModelSystem. The ModelSystem supplies its default experimental frame to the Experiment. As long as it resides in the Experiment (as Protocol instances, see figure 4), the simulation can be restarted quickly. All definitions that constitute the experimental frame can be overruled interactively during the simulation experiment.

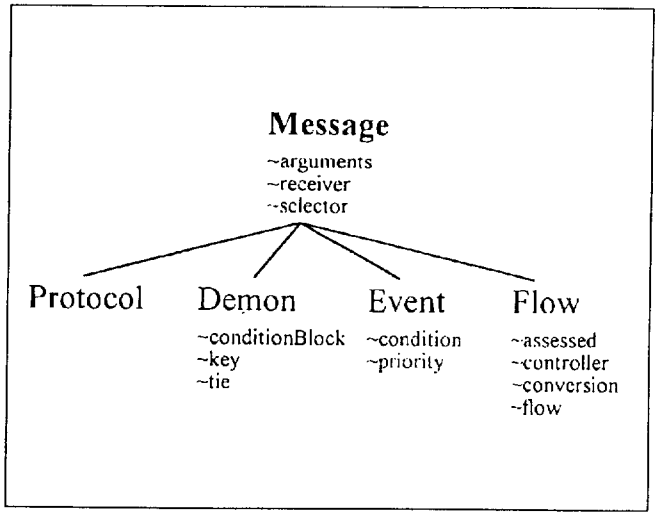


Figure 4. Subclasses of the standard Message class are used for delayed or repeated execution of code.

The Experiment connects a Simulator to the simulated system. The Simulator maintains the simulation clock and receives all notifications of time-based activations (Events, see figure 4). Notifications sent by ModelComponents are channelled through the Simulator class object, to the currently active Simulator. The Simulator adds these events to its future events list, with their time label as a key for sorting.

Ecological components

Ecological components are subclasses of Actor (figure 5). Individuals are instances of subclasses of Inhabitant; their environment is represented by instances of subclasses of Space. An individual-based simulation can, however, also be performed with Spaces alone, as will be shown in the host-parasitoid example.

Both Inhabitants and Spaces are characterized by their ability to carry out tasks, i.e., to simulate behavior. Basically, they appear in the simulation after receiving the message 'startUp', and disappear after receiving 'finishUp'. More complex behavior is simulated by executing methods representing simple actions. A few examples are listed in table 1. In general, the description of an Inhabitant's behavior should be self-contained, avoiding dependencies on other components' code that would complicate the reuse of the single component in other models.

Objects embodying the environment function as containers of a collection of Inhabitants. In addition, they may display their own behavior - although in many simulations this will not be the case. Predefined methods, as building blocks for such behavior, refer mainly to the dynamical creation of new Spaces,

Table 1. Some of the methods provided in Inhabitant (upper) and Space (lower), that can be used as building-blocks for behavior. Self refers to the object instance that is executing the method. An important task of a spatial unit may be to collect statistics on the inhabitants it contains (bottom).

die	*remove self from the simulation*
goto: aSpace	*make self an inhabitant of aSpace*
kill: anInhabitant	*remove anInhabitant from the simulation*
lookAround	*answer the inhabitants in self's Space*
lookUp	*idem, in self's superSpace*
moveRandom	*self go to a random Space*
moveRandomNeighbor	*self go to a random neighbor Space*
reproduce	*create a new inhabitant of self's type*
addLevelWithDim: aPoint	*add a new grid of Spaces below self*
addNeighbor: aDirection	*add a new Space as a neighbor in aDirection*
addSubSpaceAt: aPoint	*add a single Space below self*
averageProperty: aSymbol inhabitantType: aString	*average the values for property aSymbol over all inhabitants with class name aString*

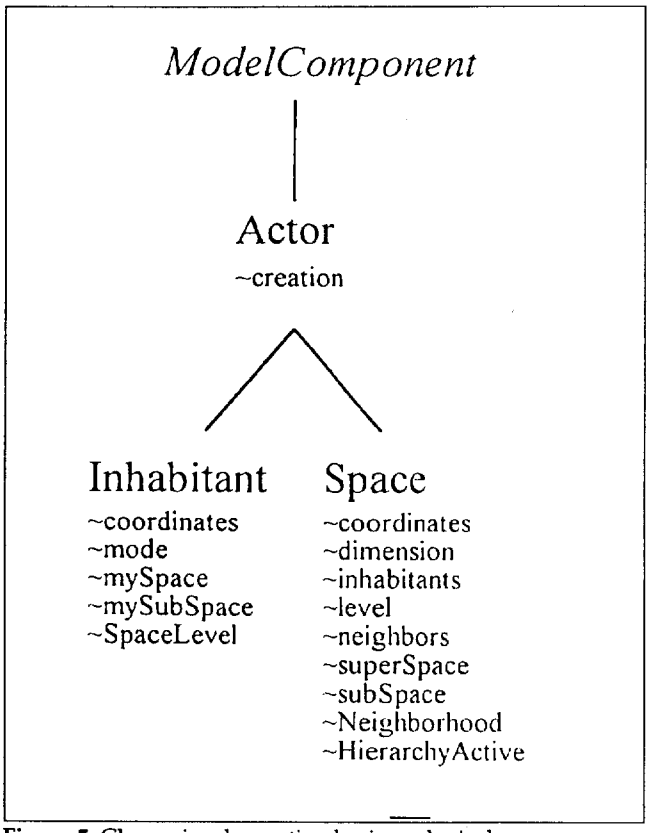


Figure 5. Classes implementing basic ecological components.

connections, and layers. Subclasses of Space can define additional properties, and events acting on these properties.

Spaces are arranged in a hierarchy by default, with large-scale elements containing small-scale ones. This feature allows simulations to involve different spatial scales. As an option, the neighborhood of each Space may either consist of 4 or 8 neighboring Spaces.

Individuals, Populations & Concentrations

EcoTalk permits a combined individual- and population/concentration-based approach. Components whose continuous dynamics are simulated, may represent individuals or aggregated units like populations. Within a discrete-event world view, continuous processes are approximated by creating a chain of events parallel to the other events related to the specific component. These events update the state of the continuous variable. This may suffice to represent relatively autonomous processes occurring inside individuals, e.g. respiration or food digestion.

For fixed interaction links between population units, EcoTalk provides another representational formalism: a System Dynamics approach (Cellier, 1991, Chapter 11). This formalism is appropriate for the specific model when a continuous-change world view dominates, and events disrupting the interaction network are scarce. A

pools-flows metaphor is applied: the state-variables are pools in a network, connected by material, energy or information flows. As a consequence, the object-oriented implementation presented in this paper is based on the classes Network, Pool and Flow (figures 4 and 6).

Each Pool represents a single quantitative property. Flows link the Pools to one another, creating a Network. This Network repeatedly schedules its own activation. Upon activation, the Network in its turn activates all Pools. The Pools then calculate their state-change according to a forward Euler integration, and set their new value. A Network is made up of distributed integrators, operating on the timestep defined by the Network. More than one Network may be present, each with its own characteristic stepsize.

Each Pool only exists in relation to, and can only be created and destroyed by a CPInhabitant. CPInhabitant (figure 6) is a subclass of Inhabitant with the ability to manipulate such internal Pools. Like Inhabitants, CPInhabitants behave event-wise. Certain events affect their Pools, e.g. when they lead to creation, destruction or modification of the interaction network in which the Pools are engaged. Conversely, the continuous subsystem represented by the Pools may affect the dynamics of the discrete system, when thresholds in Pool values are defined, that, when surpassed, trigger events in the containing CPInhabitant.

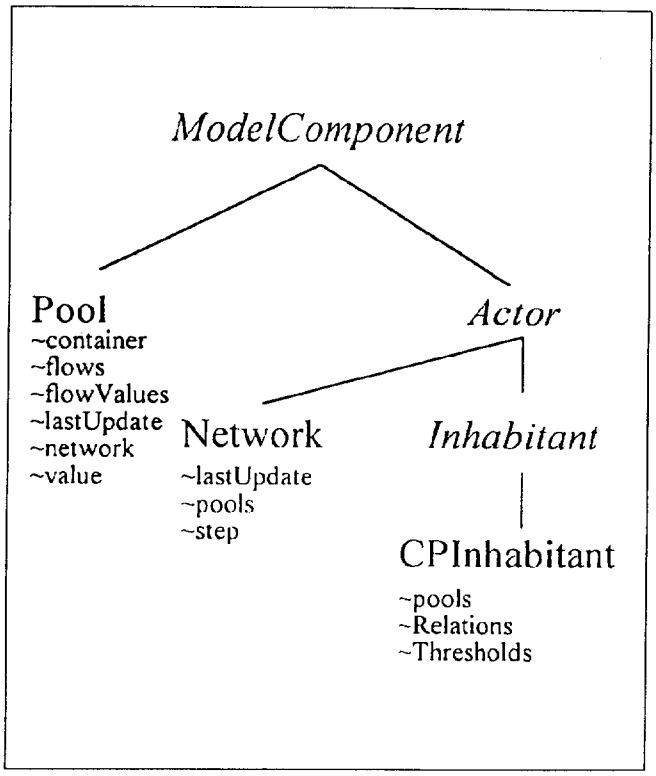


Figure 6. Classes implementing continuous system dynamics.

Collecting Data

The task of gathering information is separated from the behavior of model components. Specialized objects collect the data, operating alongside the regular components. In this way, flexibility and conceptual clarity are enhanced. Data collectors can be created any time during an experiment. Their behavior may be as complex as that of Inhabitants - or even more complex (like the observers in Hogeweg & Hesper, 1981). EcoTalk provides two basic types (figure 7).

Samplers collect the states of model components at fixed, predefined, time intervals. Observers serve the same purpose, but are not activated on a time base. They collect information after being activated by state-changes in the objects they observe. To this end, a mechanism for access-based activation (figure 4) is implemented, based on the demon concept (Murata & Kusumoto, 1989).

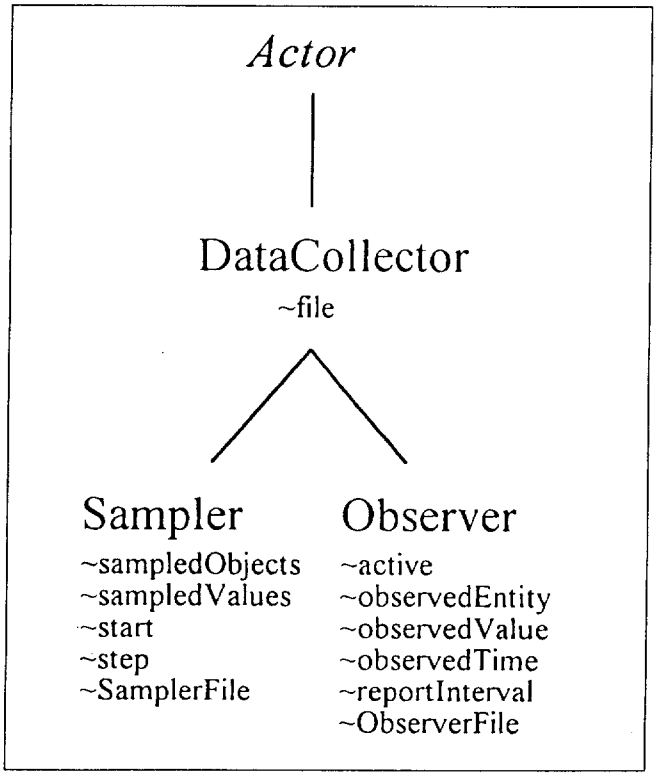


Figure 7. Data collecting classes.

EcoTalk Interface

The interface for EcoTalk was pieced together from the GUI building blocks offered by Smalltalk. Its windows are related to the different EcoTalk levels (figure 1) and serve the purposes of (1) exerting user control, (2) browsing the simulated system and (3) depicting the collected data.

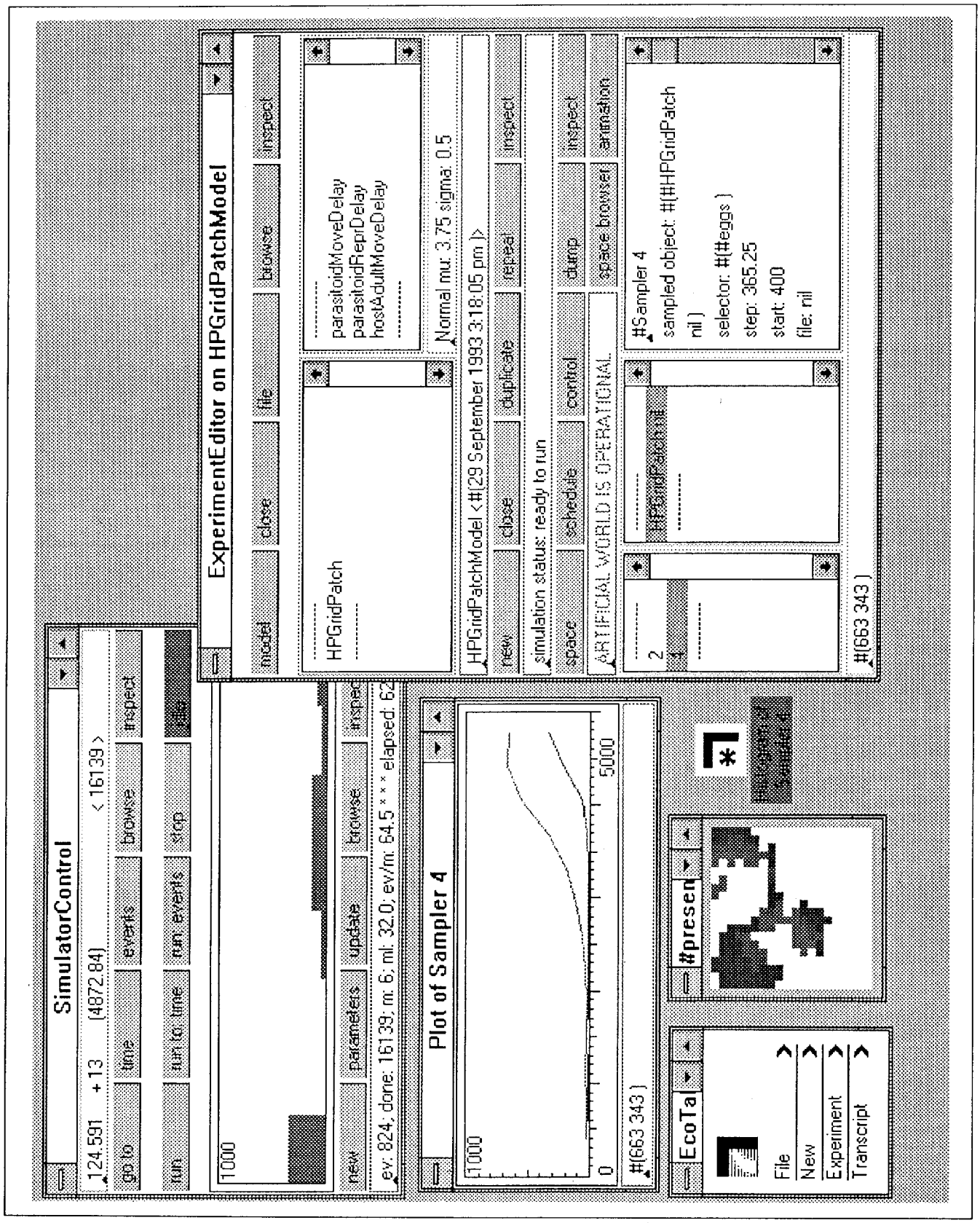


Figure 8. Appearance of some EcoTalk windows in ObjectWorks/Smalltalk

A text window, EcoTalkTranscript, is defined as a global variable. All other EcoTalk components may send information concerning their status to this window. Items selected from the EcoTalkTranscript menus lead to the creation of new ModelComponents and ModelSystems. For a new series of experiments on an existing model, an ExperimentEditor is opened.

The ExperimentEditor controls all major steps involved in a simulation. It has an Experiment as one of its instance variables. This Experiment is manipulated via the menus of the ExperimentEditor. To be more specific (figure 8), the menu items refer to high-level handling of the Experiment, editing parameters and data-collectors, executing the experiment step by step, inspecting the control structure objects, and opening views on the dynamic state of the simulation. An ExperimentEditor consists of subpanes, showing lists of the involved parameters and data-collectors. Parameters and data-collectors can be defined interactively, in dialog windows. When running the simulation in the Smalltalk development environment, source code browsers can be opened on the model components. The code can be inspected, changed and saved. An interrupted simulation may thus resume with modified component behavior.

Once the Experiment has been edited and the simulation is ready to start, a SimulatorControl window is opened from the ExperimentEditor. It offers refined control over simulation execution. A SimulatorControl consists of subpanes displaying event queue status, and depicting the event distribution. The state of the simulated system can be inspected at any moment. This is achieved by browsers that open a view on the constellation of spatial elements, and assist in navigating through space. They either apply hierarchical relationships (SpaceHierarchyBrowser) or neighborhood relationships (NeighborhoodBrowser) as a criterion. Both show the state of the system by means of lists of spatial elements and their inhabitants, as well as graphically by means of bitmaps displaying the spatial distribution of (user-defined) states. They make it possible for the user to intervene directly in the simulation and change its course of events.

Finally, windows are provided to display the data collected by Samplers and Observers dynamically, in tables, plots or histograms. EcoTalk features also an animation of the states of the spatial elements.

How to use EcoTalk?

An EcoTalk model is constructed and tested in the Smalltalk development environment. In model building, the EcoTalk kernel classes are treated as abstract classes, and not modified. Thus, for each new model a subclass is added to ModelSystem, and for each new component a subclass is added to the ModelComponent branch. In a series of experiments, these classes are tested thoroughly using Smalltalk debugging tools, and further refined. To construct an external model base, the new model classes can be written to ASCII files (figure 9: 1) or transferred to binary DLL's (figure 9: 3). By trimming the development classes from the environment, a runtime executable Smalltalk is created (figure 9: 2), containing Smalltalk and EcoTalk kernel classes plus one or more models.

Extensive analysis of model system behavior can be performed in such a runtime system. Each experiment starts with the default experimental frame, laid down in the ModelSystem's source code. Interactively, the experiment is tailored by the user, supplying other parameter values, creating new data collectors, opening different windows, and so on. Only the behavior of the ModelComponents cannot be changed (in runtime systems). Using binary object storage facilities, complete simulations can be interrupted and stored, for use in different sessions. They may constitute a simulation base (figure 9: 4). Collected data are exported to external files, or exchanged with other data-processing packages (e.g. spreadsheets) through Dynamic Data Exchange or clipboard facilities.

ILLUSTRATIONS

Host-Parasitoid Spatial Dynamics

Host-parasitoid systems describe a basic interaction in ecology, for animals (hosts) parasitized by animals of similar size. Analysis of simple mathematical models has shown that in homogeneous environments, these interactions, like comparable predator-prey interactions, may result in complex population dynamics. In spatial systems a new level of complexity is added. Local populations can go extinct, while uninhabited areas can become colonized. Simulations have shown that extinction and colonization may balance one another, resulting in a global persistence of both populations. Hassell et al. (1991) demonstrated that diffusion-like (local) dispersal in such systems may produce persisting spatial patterns, including high-density waves travelling or spiralling through the environment. The results were obtained in simulations of difference equation and cellular automata systems.

An EcoTalk model is constructed to analyze the spatial dynamics of host-parasitoid systems (Baveco & Lingeman, 1992). It simulates locally interacting individuals, with local dispersal. Unlike previous work, it makes it possible to relate the spatial patterns of distribution to the detailed behavior of individuals, e.g. individuals may move directionally, motivated by the

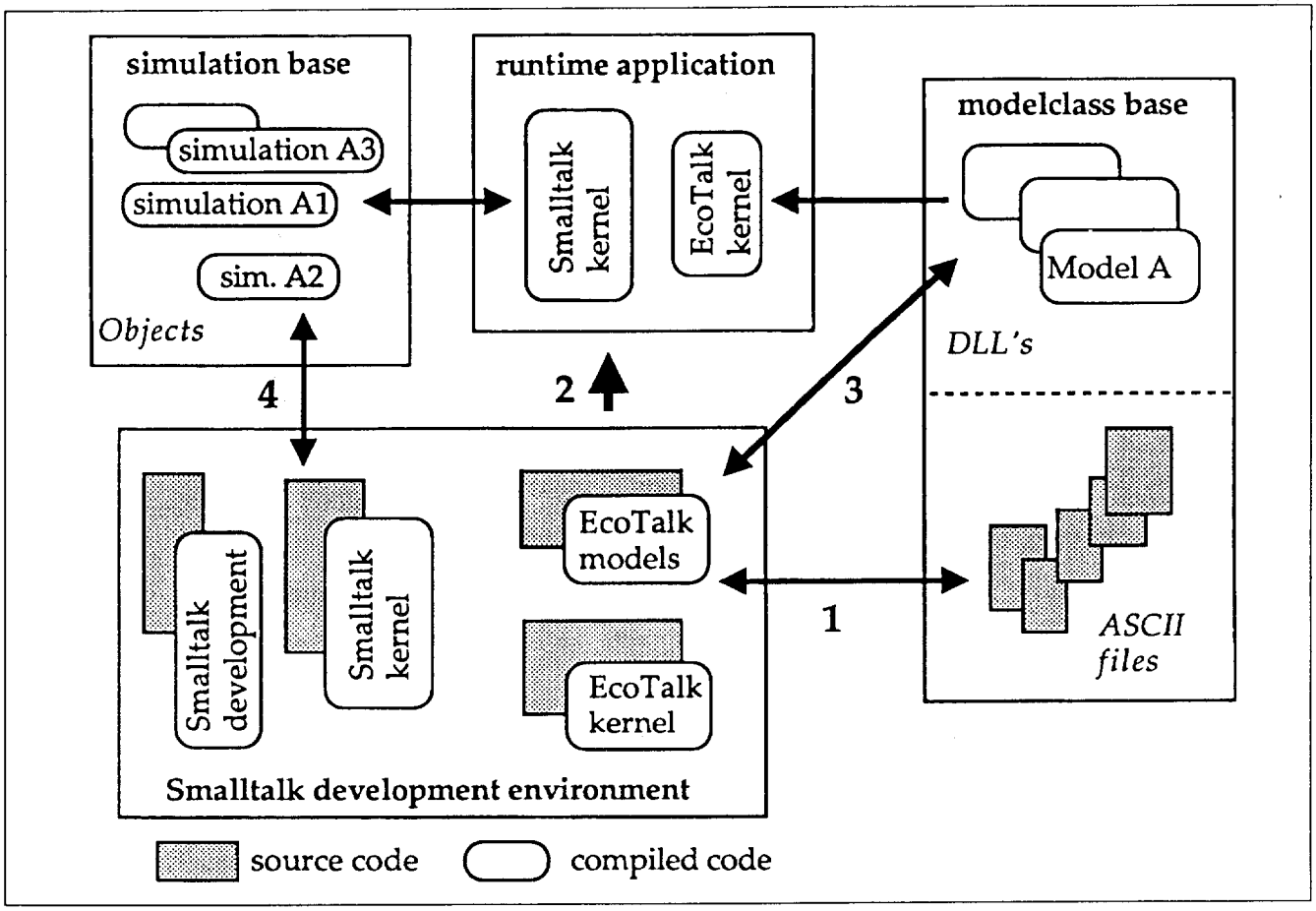


Figure 9. Facilities for orgazanizing models and simulations, in modern Smalltalk environments.

status of their immediate environment, their accumulated experiences or other strictly individual parameters.

Two versions of the model have been developed. Initially, each individual and each location was represented by an object in the program. To improve simulation performance, in an alternative model the system was represented exclusively by locations (patches). The state of a patch was defined as the number of hosts and parasitoids it contained. In the individual-based version, events occurred to individuals. Many of these events changed the properties of a spatial unit, e.g. dispersal, reproduction and death changed the number of Inhabitant objects in a patch. In the patch-based model, the same type of events occurred. However, these events only changed the state of the patches.

Fish Interactions

In fish communities numerous examples are found of organisms with a position in the foodweb that is changing during their life cycle. At different stages in their ontogeny, they may be competitors, predators and/or prey. The interactions between species are a function of body size, age and the trophic structure of the community.

Interactions between two species, taking into account the age-structure of the populations, have been simulated with ordinary differential equation models (Post & Rudstam, 1991). In such an approach, an average body size is assumed to apply to all individuals in an age-class. However, as body size is a critical parameter in determining the interactions, in a more detailed investigation of the regulatory mechanisms, the size distribution within an age-class should be taken into account. In general, such size-structured populations are modeled by partial differential equations (Metz & Diekmann, 1986). In case of small populations, certain stochastic effects, or incomplete mixing, the conditions for a PDE approach are not met. As an alternative, each individual in the population is followed separately. Experiences with such individual-based simulations are limited to single-species models (Adams & DeAngelis, 1987).

Currently, an EcoTalk model is being developed for two interacting fish populations. Based on individuals, small-scale differences in body size are taken into

account. This approach allows a detailed investigation of the role of size-based compensatory mechanisms in the age classes exhibiting the highest plasticity in individual growth (i.e. the juvenile stadia). A related research topic refers to an investigation of the effects of stochasticity and individuality when incorporated into a model of size-based interactions between a consumer-species and its food. The model formulations in question range from a set of differential equations (all individuals equal) to a discrete-event individual-by-individual model.

PERFORMANCE

For the host-parasitoid application described above the performance in terms of processed events per second is estimated. An individual-based and a patch-based version are compared, running under DOS 4.0 with Windows 3.0 (Smalltalk/V Windows 1.1), and under OS/2 1.3 with Presentation Manager (Smalltalk/V PM 1.3). The results, averaged over three runs, are depicted in figure 10 and table 2. All runs started with a dozen initial events. In the course of the simulation the number of events in the event queue fluctuated heavily with peak values steadily increasing up to (at the end of the run) approximately 4000 to 5000. Figure 10 shows that no serious degradation of event queue performance was observed with increasing queue size. Table 2 indicates the performance gain by switching from an individual-based to a patch-based representation (1.45 for Smalltalk/V PM and 1.27 for Smalltalk/V WIN). In Smalltalk/V PM the simulations were executed almost twice as fast, on the same platform.

DISCUSSION

Illustrations

In developing both applications, we experienced the benefits of object-orientation. New models were constructed efficiently, and with relatively minor efforts. The interactive nature of Smalltalk, and its flexibility, sped up model development and greatly encouraged an explorative modeling approach, testing different model designs (Baveco & Lingeman, 1992). Investigating a specific model, the EcoTalk interface (added at a later stage) proved very helpful in tailoring experimental design.

The host-parasitoid model indicates that EcoTalk can be used profitably for medium-sized individual-based applications. On the particular platform, simulations of up to approximately 10,000 individuals were feasible (Baveco & Lingeman, 1992). However, for the model system in question this proved inadequate to investigate large-scale spatial pattern formation.

The fish population application includes both small- and medium-sized individual-based models. Preliminary results (Scheffer & Baveco, in prep.) indicate that, although some of the emerging population phenomena could have been produced in differential-equation models as well, the interpretation of the underlying mechanisms is more straightforward in an individual-based setting.

Expressiveness

Experiences with EcoTalk are limited to simulations involving numerous components with a relatively simple behavior (i.e. less than a dozen different events

Table 2. Performance of EcoTalk Simulations, in terms of average event duration and processed events per second.

	Smalltalk/V PM		Smalltalk/V WIN	
	individuals	patches	individuals	patches
seconds/event	0.07	0.048	0.119	0.094
events/second	14.3	20.8	8.5	10.8

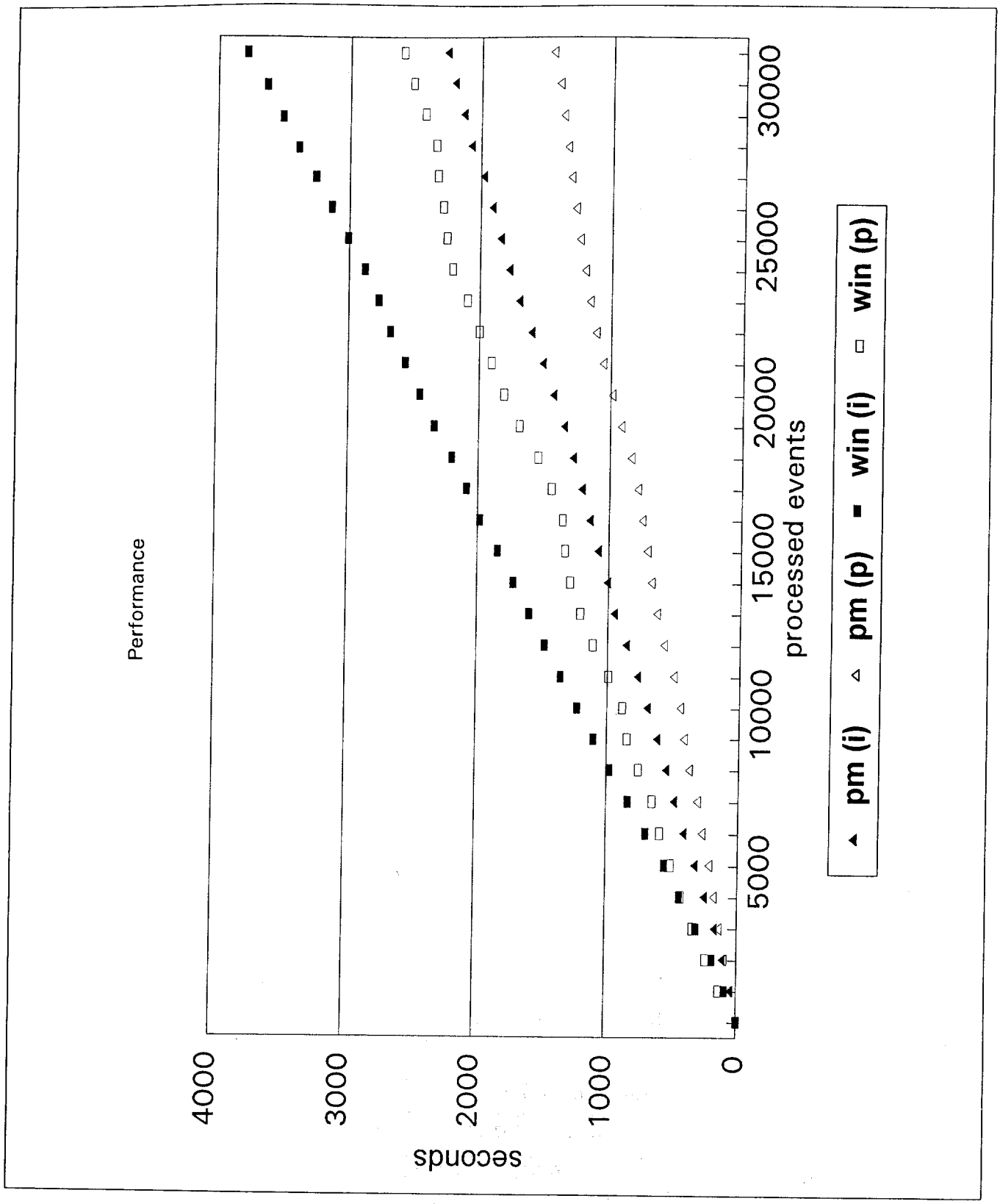


Figure 10. Comparison of the performance of two EcoTalk simulation models, under different operating systems. The models are an individual-based (i) and a patch-based (p) representation of a host-parasitoid system. The simulations are performed under the Windows (win) and Presentation Manager (pm) GUI. The data represent the average of three simulation runs. For a discussion of the results, see text.

for each component). For these models the applied event scheduling approach is appropriate. Whether this will also be the case for more complex interactions remains to be seen.

In the fish population model, continuous processes, fish respiration and zooplankton growth, are simulated under a discrete-event timing regime. Their inclusion is straightforward. However, ad hoc assumptions are required to handle the state-variables that are subject to change by both continuous and discrete processes. For instance, zooplankton growth occurred on a fixed timestep base, while zooplankton consumption by fish occurred on an event base. In this case, the consumption during the timestep was summed and at the end of the timestep balanced with the increase by growth, to yield the next biomass value.

In EcoTalk a discrete environment is assumed, made up of spatial units. Inside a unit, homogeneity is assumed. For individuals whose behavior is guided by fine-grained local information (e.g. the proximity of other individuals), this grid-like representation may be too coarse. Addition of spatial units with, internally, a continuous coordinate system, and the associated inhabitants moving inside these units, is under consideration.

The use of classes as object templates in Smalltalk poses restrictions on the representation of evolutionary processes. It is easy to simulate genetic change as long as it is confined to the values of instance variables of objects and their offspring. The methods defined in a class, however, apply equally to all instances. Solutions to this problem of instance-specific methods, however, are described by Beck (1993).

Performance

The processing time per event, as measured above, allows estimation of the feasibility of a specific modeling enterprise in advance. Better performance, clearly, is to be expected on faster platforms, i.e. 486-based PC's and workstations. We improved performance by selectively optimizing intensively used object methods. With the host-parasitoid model, a substantial performance gain resulted from switching to another system representation. A trade-off is noted between flexibility and performance. It is in the nature of object-oriented programming to create subclasses that are tuned to perform certain tasks, at the expense of the flexibility displayed by their parent class. For instance, in EcoTalk a Space object contains its subspaces (pointers) as values in a flexible Dictionary, with coordinates as their look-up key. In the host-parasitoid simulations, optimized subclasses of Space were used, with subspaces contained in a faster accessible, but less flexible, Array.

Smalltalk and Simulation

For the purpose of individual-based modeling and simulation, Smalltalk fell short in the following aspects: (1) an efficient general-purpose mechanism for access-based activation is absent, (2) multiple-inheritance is not supported. The demons constructed in EcoTalk to broadcast state-changes, served well from a practical point of view. Conceptually they do not satisfy, as the object to which demons may be tied still needs to take explicit action in order to activate its demons, upon an internal state-change. The same objection applies to the dependency-mechanism that is a standard Smalltalk feature (each object may have dependents). The dependency-mechanism is heavily used by and interwoven with the Model-View-Controller implementation, and therefore not used to mimic access-based activation in EcoTalk. Multiple-inheritance, i.e. the ability to inherit from more than one superclass, would make it possible to break down the ModelComponent branch into several small hierarchies, each embodying a single concept. Instead, much time was invested in finding an acceptable single-inheritance structure, based on considerations of simulation approach, biological level of organization, and task in simulation.

With EcoTalk, a modeling system is constructed inside a programming system. The modeling system itself can be modified as easily as the user-defined models. On the one hand, this appears an advantage: the modeling system may benefit and mature from the applications developed in it. On the other hand, unlimited access to all parts of the system for all users is dangerous (system crashes may easily occur), and undesirable because modifying essential features will limit the possibilities of model interchange between modelers. We found a workable solution in the convention to treat basic EcoTalk classes as abstract classes. In developing a new model, these classes are not used; instead subclasses are created. The basic classes are modified only by optimizing methods internally (without affecting the object's interface) or adding new methods. To Smalltalk-illiterate end-users, models are supplied as runtime EcoTalk systems, protected against any modification, but still allowing all kinds of experimentation.

A major asset of the Smalltalk environment is that it provides for a Personal Simulation System, constantly evolving and accumulating the experiences of the modeler. The simulation system presented in this paper can, thanks to its object-orientation, grow in different directions. At the end of the EcoTalk branches, new model classes appear, inheriting functionality from the EcoTalk kernel classes and ecological detail from each other. The kernel classes in turn evolve to become even more fine-tuned to their tasks, while now and then they

shoot out branches, forming a substrate for new types of models.

Acknowledgements

We would like to express our gratitude towards R. Lingeman, whose energetic support made this research possible. We thank M. Scheffer for his willingness to act as the unsuspecting user on whom the ever-changing user-interface could be tested.

References

Adams, S.M. and D.L. DeAngelis. 1987. Indirect effects of early bass-shad interactions on predator population structure and food web dynamics. In: W.C. Kerfoot & A. Sih (Eds.), *Predation in Aquatic Ecosystems*, University Press of New England, Hanover, NH, pp. 103-117.

Baveco, J.M. and R. Lingeman. 1992. An Object-Oriented Tool for Individual-Oriented Simulation: Host-Parasitoid System Application. *Ecol. Modelling* 61: 267-286.

Beck, K. 1993. Instance specific behavior: how and why. *The Smalltalk Report* 2 (6): 13-21.

Cellier, F.E. 1991. *Continuous System Modeling*. Springer Verlag.

Giron-Sierra, J.M. and J.A. Gomez-Pulido. 1991. Doing Object-Oriented Simulations: Advantages, New Development Tools. *Proceedings 24th Annual Simulation Symposium*, New Orleans, Lousiana, April 1-5, 1991, pp. 177-184.

Goldberg, A. 1990. Information Models, Views, and Controllers. *Dr. Dobbs Journal* 15 (7): 54-61.

Goldberg, A. and D. Robson. 1983. *Smalltalk-80: The Language and its Implementation*. Addison-Wesley Publishing Co., Reading, MA.

Hassell, M.P., H.N. Comins and R.M. May. 1991. Spatial structure and chaos in insect population dynamics. *Nature* 353: 255-258.

Hogeweg, P. and B. Hesper. 1981. On the role of OBSERVERS in large scale systems. *UKSC Conference on Computer SIMULATION*, Westbury House, Harrogate, pp. 420-425.

Hogeweg, P. and B. Hesper. 1990. Individual-Oriented Modelling in Ecology. *Math. Comput. Modelling* 13(6): 83-91.

Huston, M., D. DeAngelis and W. Post. 1988. New Computer Models Unify Ecological Theory. Computer simulation shows that many ecological patterns can be explained by interactions among individual organisms. *BioScience* 38(10): 682-691.

Kirkpatrick, S. and E. Stoll. 1981. A Very Fast Shift-Register Sequence Random Number Generator. *Journal of Computational Physics* 40.

Kreutzer, W. 1987. A Modeller's Workbench: Experiments in Object-Oriented Simulation Programming. In: J. Bézivin, J.-M. Hullot, P. Cointe and H. Lieberman (Eds.), *Proceedings European Conference on Object-Oriented Programming '87*, pp. 203-212.

Larkin, T.S., R.I. Carruthers and R.S. Soper. 1988. Simulation and object-oriented programming: the development of SERB. *Simulation* 52(3): 93-100.

Lhotka, L. 1991. Object-oriented methodology in the field of aquatic ecosystem modelling. In: J. Bézivin and B. Meyer (Eds.), *TOOLS 4. Technology of Object Oriented Languages and Systems*, Prentice Hall, Hemel Hempstead (UK), pp. 309-317.

Metz, J.A.J. and O.Diekmann (Editors). 1986. *The Dynamics of Physiologically Structured Populations*. Lecture Notes in Biomathematics, 68. Springer, Berlin.

Meulen, P.S van der. 1989. Development of an Interactive Simulator in Smalltalk. *Journal of Object-Oriented Programming*, January/February: 28-51.

Murata, M. and K. Kusumoto. 1989. Daemons, another way of invoking methods. *Journal of Object-Oriented Programming*, July/August: 8-12.

Ören, T.I. and B.P. Zeigler. 1979. Concepts for advanced simulation methodologies. *SIMULATION* 32(3): 69-82.

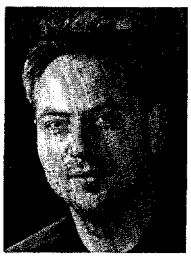
Pinson, L.J. and R.S. Wiener. 1988. *An Introduction to Object-Oriented Programming and Smalltalk*. Addison-Wesley Publishing Co., Reading, MA.

Post, J.R. and L.G. Rudstam. 1991. Fisheries Management and the Interactive Dynamics of Walleye and Perch Populations. In: J.F. Kitchell (Ed.), *Food Web Management: A Case Study of Lake Mendota, Wisconsin*. Springer-Verlag.

Rönngren, R., J. Riboe and R. Ayani. 1991. Lazy Queue: An Efficient Implementation of the Pending-event Set. *Proceedings 24th Annual Simulation Symposium*, New Orleans, April 1-5, 1991, pp. 194-204.

Saarenmaa, H., N.D. Stone, L.J. Folse, J.M. Packard, W.E. Grant, M.E. Makela and R.N. Coulson. 1988. An artificial intelligence modelling approach to simulating animal/habitat interactions. *Ecol. Modelling* 44: 125-141.

Zeigler, B.P. 1990. *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*. Academic Press, San Diego, CA.



HANS M. BAVECO undertook the development of EcoTalk as part of his PhD project on ecological simulation modeling at the Department of Pure and Applied Ecology of the University of Amsterdam. In this project, developments in modeling and simulation and AI (e.g. object-oriented programming and knowledge-based systems) were evaluated on their perspectives for application to ecological problems. His background includes a MS in ecology (1985) from the University of Utrecht, with a specialization in population-dynamics, and a position at Delft Hydraulics where he participated in a large-scale salt-water lake ecosystem modeling project (including nutrient-algae and benthic macrofauna submodels). His present work at the Institute for Forestry and Nature Research concerns the assessment of ecotoxicological effects of pesticides, by means of individual-based energy-budget models. From these models, population consequences of the application of pesticides are deduced.

ARNOLD W.M. SMEULDERS is Professor of Biology, Chair of Biological Informatics and Associate Professor of Mathematics and Computer Science, Chair of Image Information Systems, both at the University of Amsterdam. His research interest is mainly in computer vision, sensor information system design and simulation as a tool for experimentation/field research in the real world.