ELSEVIER

Ecological Modelling 82 (1995) 265–276

ECOLOGICAL MODELLING

Hierarchical object-oriented design of a concurrent, individual-based, model of a pelagic Tunicate bloom

Ph. Laval [*]

Observatoire Océanologique, URA 716 du C.N.R.S. Station zoologique, B.P. 28, 06230 Villefranche-sur-Mer, France

Received 18 January 1994; accepted 10 May 1994

Abstract

This paper stresses the importance of software engineering techniques for ecological modeling. The "object" paradigm has changed the way ecologists now view computer modeling, but the power and flexibility of the object approach call for a rigorous methodology. The interest of the HOOD method (first developed for the space industry) is shown on an ecological example, which uses Artificial Life techniques to simulate the development of a Tunicate bloom.

Keywords: Individual-based models; Object-oriented models; Zooplankton

1. Introduction

Ecology is the study of complex biological systems in relation to the environment. Software engineering is the management of complex arrangements of pieces of software in order to model an application domain. Because of the rapid increase in performance of hardware, and of the flexibility of new computer languages, software engineers designing more and more demanding applications have been readily confronted with complexity. What has been called the "software crisis" has led to the development of methods aimed at managing complexity (Booch, 1991). These methods were necessary considering that, for example, the on-board software for a space shuttle comprises 500 000 lines of code, and 1 700 000 for the on-ground environment (Myers, 1988). In the ecological domain, where the underlying structures are in most cases probably more complex than the ones found in a space shuttle, the tendency has first been to drastically simplify the real world, hoping that it would then be amenable to a mathematical analysis. There is some justification to searching for simple explanations, and trying to include in a model every component of a system may quickly lead to obscurity (Berryman, 1993). However, more and more voices are now pointing out that complex systems are of another order than complicated systems (see for example Jørgensen et al., 1992). In dealing with complex models, ecological modelers may profit from software engineering methodologies: in this discipline methods now exist which permit us to abstract the main proper-

[*] Corresponding author.

ties of a system, to decompose a system in subsystems, or to find the key components at a given level of organization. Of course the major difference between software engineering and ecological modeling is that in the latter case the organization, and sometimes even the composition of the real world is largely unknown. In both areas, however, there is a need to order structural or functional complexity. This need has been successfully answered in some large recent software projects.

Drawing a parallel between the modeling activity of software engineers and ecologists, this article will first stress out the importance of the "object revolution", and will point up the interest of a hierarchical decomposition for breaking down ecological models. In contrast to the object-oriented approach using inheritance which is frequently utilized, an object-based approach using the Ada language will be advocated to lay the foundations of well-structured ecological models. An application of the HOOD method to design a simulation of the development of a Tunicate bloom will be presented, using self-reproducing concurrent individuals to yield an individual-based simulation.

2. Software engineering methods and ecological modeling

One of the prominent trends of modern software engineering is the move toward an object-oriented approach. The advantages of object-oriented modeling have been recently emphasized by ecologists (Saarenmaa et al., 1988; Lhotka, 1991; Sequeira et al., 1991, Sequeira et al., 1993; Baveco and Lingeman, 1992; Liu, 1993; Makela et al., 1993; Maley and Caswell, 1993; Silvert, 1993). The change from a functional approach (i.e. equation-based models easily expressed in FORTRAN) to an object-oriented approach is a cultural revolution, which now strikes ecologists in the same manner than it stroke software engineers some years ago. This change of paradigm may appear superficially as a computer language choice. The language is, of course, only a means to express some concepts. However, if the language is not expressive enough, some concepts cannot be expressed at all. There is now a general agreement in the software community that object-oriented methods provide a better way to model a real domain.

3. Object-oriented design

Earlier methods of software design, such as SA/SD (Yourdon and Constantine, 1978; DeMarco, 1979) or JSD (Jackson, 1983), were centered upon a functional decomposition of the domain. They used concepts somewhat familiar to ecologists, like data flow diagrams or state transition diagrams. They were not formally used for designing ecological models because these models were hardly complicated enough. With the advent of object-oriented models, where real world entities or conceptual entities are represented by "objects", a methodology is indispensable to properly organize numerous objects and their relationships.

Several text-books have been published on object-oriented methodologies (Shlaer and Mellor, 1988,1992; Wirfs-Brock et al., 1990; Booch, 1991; Coad and Yourdon, 1991; Rumbaugh et al., 1991). A comparison of a great number of object-oriented methods can be found in de Champeaux and Faure (1992). A more fundamental framework is established by Batory and O'Malley (1992); they stress the power of hierarchical layered design for modeling complex systems, and state (p. 381) "Astonishingly, the idea of stratified designs and layered system is virtually absent in contemporary object-oriented literature". However, these principles form the basis of the HOOD method.

4. The HOOD method

HOOD (Hierarchical Object-Oriented Design) is the standard European Space Agency method for large software developments. It is based on a formalization of methods expressed in Seidewitz and Stark (1987) or Bailin (1989). The advantages of this method for ecological modeling are threefold:

(1) The HOOD decomposition starts from a global, abstract view of the whole system (the "root object"), which is broken down into component objects, which are in turn decomposed until "terminal objects" are found. This top-down decomposition is known to be a convenient way to disclose complex structures (Rasmussen, 1985).

(2) HOOD produces structures which can be easily expressed in Ada. The Ada language is appealing to ecological modelers (Murray, 1990). Besides its powerful and rigorous potentialities for numeric calculations (greatly superior to FORTRAN), Ada is designed around sound software engineering principles, including object-based features and built-in concurrency. It is safer than C++ (Reed and Wyant, 1992; Jørgensen, 1993).

(3) HOOD diagrams are designed for conveying the structure of a complex system during the specification phase, at any level of organization, in a way understandable both to the programmers and the contractor. These diagrams are therefore helpful for displaying an ecological model with little references to a programming language.

5. Object-oriented and object-based modelling

In its present form, Ada does not include inheritance, so that it is object-based rather than object-oriented (Chin and Chanson, 1991). Inheritance was not incorporated in Ada because at the time of its first normalization (1983) this concept was not thoroughly explored, and it was anticipated that it may lead to misuses. Now after long debates, the next major revision of Ada, presently known as Ada 9X (Barnes, 1993) will include only a controlled form of inheritance ("programming by extension"). This is because an indiscriminate use of inheritance may rapidly lead to unsurmountable problems in designing large systems. "True" object-oriented methods (i.e. using inheritance, like the method of Rumbaugh et al., 1991) induce designs built upon an entity-relationship model. This is not the case for the Booch (1991) method or the HOOD method.

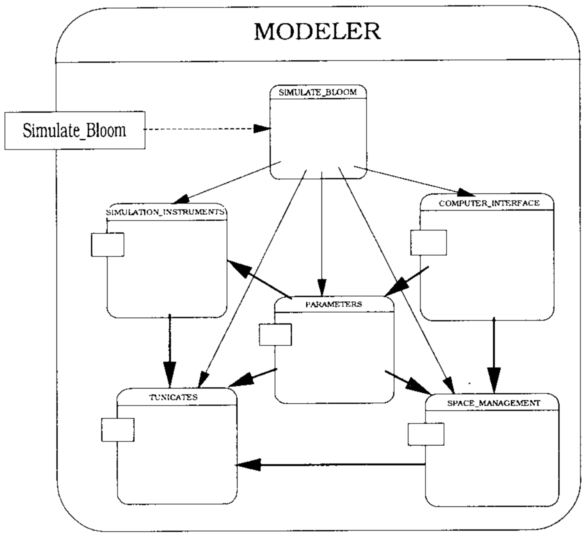


Fig. 1. Simplified HOOD decomposition diagram of the root object, Modeler. The operation Simulate_Bloom in the interface box on the left edge triggers an Operation Control Structure with the same name (as indicated by the broken arrow). This OPCS acts upon 5 child objects. An arrow directed from a child object, for example Simulation_Instruments, to another child object like Tunicates, indicates that the latter uses the former. Data flows and exception omitted, as well as the environment objects (utilities).

An illuminating discussion of the implications of inheritance may be found in Rosen (1992). This analysis shows the advantages of an object-based decomposition for designing complex systems. An object-based hierarchical design is appropriate for setting up the foundations of a robust ecological model, which may be safely augmented when additional knowledge is available.

6. Outline of the HOOD method

The HOOD Reference Manual (Delatte et al., 1993) gives a rather formal view of the method, in the form of definitions and rules destined to be implemented in software tools. A more practical exposition may be found in Lai (1991). An explanation of a few terms may be helpful for the remaining of this article.

In a HOOD diagram like the one in Fig. 1, an object at level n is represented by a named rectangle with rounded corners. This object furnishes a number of services (or operations), through an interface drawn as a box on the left edge of the diagram. These operations are implemented at level $n + 1$ by a small number of child objects, included in the parent object. Objects are passive (they execute sequentially their operations) or active (the operation they execute depends on their internal state). The operations of passive objects are coded in an Operation Control Structure (OPCS), while the operations of active ones are mediated through an Object Control Structure (OBCS). In a parent object including the child objects A and B, "B uses A" is depicted in the diagram by an arrow flowing from A to B. In the corresponding Ada code, the parent object package is "with Object_A, Object_B;", and Object_B (which is "with Object_A;") may request Object_A.Operation_2, in doted notation. Operations are implemented by Ada procedures, functions or task entries. An OPCS is represented by a rounded rectangle without interface box, and is implemented by the body of a procedure or function. An OBCS is implemented by an OBCS package encapsulating a task.

A HOOD decomposition begins by determining the top level object of the model. This root object must capture an abstract view of the whole model. This step may superficially appear trivial, but is in fact very difficult. The later a missing object is re-introduced in a HOOD diagram, the costlier the consequences. This may lead to a complete redesign (and recoding) of the model. Once identified, the operations at the parent level are implemented by operations in child objects included in the parent. This child objects will be similarly decomposed at the next design level. This process continues until no further children are necessary, in which case a "terminal object" has been reached. Up to six levels of decomposition may be necessary to organize and document large software systems which may comprise more than 100 objects. For smaller systems, two to four levels are usually sufficient. The power of the method for describing an ecosystem may be appreciated if one considers that each level constitutes an "abstract machine", the details of which will be found in further levels.

Drawing the diagram of the root object is only possible after a domain analysis. What essential entities should pertain to our abstraction of the reality, keeping in mind that their relations with each other should be fully understood? This is not a straightforward task. The reader is referred to the above-cited literature, where cues can be found on how to find "good" objects and how to interface them; papers by Colbert (1989), Ladden (1989), Whitcomb and Clark (1989), Freitas et al. (1990), and Walters (1991) may also be useful in an object-based context. With HOOD, the Ada compiler greatly helps in detecting inconsistencies, because HOOD objects correspond to Ada packages, so that the compiler checks the "visibility" of every name in the parent–child hierarchy. Several CASE (Computer-Aided Software Engineering) tools are available, which may assist the designer in drawing interactively the diagrams, and in writing Ada code skeletons from the specifications.

7. An ecological example: The development of a Tunicate bloom

An individual-based model should be a concurrent model, where each individual behaves autonomously. Among all object languages, Ada provides the best model of concurrency (Burns et al., 1987; Shumate, 1988). Ada tasks can be successfully used to add a concurrent behavioral component to the structure of every individual (Laval, 1995). This paper shows how an artificial individual may behave autonomously, and transmit its structure to its offspring. The flexibility of the process is best viewed with animals having a complicated life cycle, like the salps.

Salps are pelagic oceanic Tunicates presenting an alternance of sexual and asexual reproduction (Alldredge and Madin, 1982). The individual issued from the egg is called an oozooid. This oozooid grows and develops a stolon, which, by asexual reproduction (segmentation), forms a chain of similar individuals, named blastozooids or aggregates. The blastozooids remain attached together by the tunic, their external envelop, so that the whole chain moves in the water. In the chain, each blastozooid matures and becomes first a female, and then a male. When female, they can be fertilized and produce an egg, which will develop into an oozooid. The potential of this reproductive cycle is apparent if one considers that an oozooid of *Salpa fusiformis*, a common species, gives birth to a chain of more than 100 blastozooids; each of these blastozooids will produce about 100 oozooids. After only 4 generations, assuming no mortality, 100^4 (one hundred millions) individuals may be produced. An oozooid produces a chain in 6 to 9 days, and the oozooids appear after 8 to 10 days (Braconnot et al., 1988). Moreover, an oozooid yields about 3 chains during its life. This explains why Tunicates may cover hundreds of km^2 of the sea (Fraser, 1961; Foxton, 1966; Berner, 1967; Madhupratap et al., 1980).

The ecological importance of Tunicates in the pelagic food chain is a motivation to model their explosive development (Andersen and Nival, 1986; Braconnot et al., 1988; Ménard et al., 1993). Here a simulation of the development of a Tunicate bloom will be undertaken along the lines of the HOOD methodology.

7.1. The root object and its decomposition

Simulating the development of a Tunicate bloom may be seen from two points of view. A "classical" modeling gives an active role to the modeler, which sets everything in the model in a deterministic way. This is a functional approach. In an object-oriented point of view, it is necessary to consider more autonomous entities. There is, of course, a modeler, but there are also some instruments needed for the simulation, like a clock; and there is the Tunicate population. In a more extensive model, other essential entities should naturally be taken into account: a nutrient pool, some phytoplanktonic or picoplanktonic species used as food by the Tunicates, a parasitoid eating the Tunicates, and so on. But one biological entity and the objects needed to carry the simulation are sufficient to give the main lines of a HOOD decomposition. This design is translated into an Ada program named CALIFE (Computational Artificial LIFE). The source code of the program is available upon request from the author. In CALIFE, the modeler does not act upon the Tunicate population (except for creating an initial oozooid). At the onset of the program, the modeler triggers a clock, and lets the Tunicate population grow. At the end of the simulation duration, the Tunicates stop (so they should have visibility over the simulation clock). The modeler is only allowed to trace the final state of each Tunicate back to the first oozooid, and to count the number of oozooids or blastozooids born, how many died, etc. That is, the modeler is an observer of the simulation. After many preliminary design drafts, the root object of the HOOD decomposition appeared to be best expressed by a Modeler object, representing an ecologist doing a simulation of the development of a population of artificial Tunicates. These autonomous self-reproducing artificial Tunicates are programmed with Ada tasks (Laval, 1995).

The artificial Tunicates are made of computer memory cells, shaped by Ada strong typing rules in order to mimic the reproductive pattern of real Tunicates. These creatures move in the video memory, which has the side effect of making them visible on the computer display. Their movements occur in some abstract Space object, but because this is a laboratory simulation, this object must map to a physical device. To comply with the constraints of this device, there should be a mediation between the Tunicate movements and the physical space. This is accomplished in a Space_Management object (at the further decomposition level, this object will include a Display_Buffer object, programmed along the lines of the Display_Buffer_Package of Cohen, 1986, pp. 788–793).

Besides the Space_Management child object, the Modeler includes four other child objects: a Parameters object, a Simulation_Instruments object, a Computer_Interface object, and of course a Tunicates object. These objects realize the simulation through an OPCS named Simulate_Bloom (Fig. 1). The Parameters object sets the parameter types and reads their initial values in a configuration file. Simulation_Instruments includes a Simulation_Clock and a Zooid_Counter, which is only used when the simulation is finished, to accumulate the numbers of zooids and their states. The Computer_Interface object furnishes some basic services to carry the simulation on a computer, such as clearing the screen or hiding the cursor, as well as setting an introductory screen. The Tunicates object, detailed in Laval (1995), represents the population issued from the first oozooid.

7.2. The spatial dimension

In the CALIFE program, Tunicate movements occur inside a rectangular grid, bounded with a frame. Each Tunicate is represented by one character, and can move in one of 8 directions. Different characters are used for live oozooids, dead oozooids, live and dead chains. A chain, constituted by identical blastozooids, is considered as a single (super)individual because all its component blastozooids move together, in the same direction, and perform the same physiological functions. At this granularity, representing every blastozooid would needlessly complicate the program. Tunicate movements are not random, but rather use an elaborated algorithm for simulating the spiral swimming of real Tunicates, and for changing the swimming direction in case of collision. Cadavers, which are an important ecological category (their decay transfers organic matter to other trophic levels), remain active until complete decomposition. A cadaver ceases to swim, but "falls" slowly downward the screen bottom. When completely decomposed, their display is erased.

At the design level, the important point is how to manage the space? Tunicates have no global spatial knowledge: they only know that a position is occupied when they bump into something. How does a real Tunicate know that there is an obstacle at the position of its next move, and reverse its swimming direction? In the open ocean, obstacles are uncommon, except when the individual are tightly packed in a dense swarm. However, this is not true of the personal computer screen, and we must cope with this situation. In the program, a Tunicate should not be allowed to move to an occupied position, so somewhere there must be a map of the space. In a concurrent program, a "monitor task" must control the use of a shared resource, permitting its access only by mutual exclusion. This approach was tried, but when several hundred individuals are simultaneously active, far too much requests are made to the monitor, which cannot service them fast enough, and the program is eventually frozen. Suppressing the monitor task has, however, only minor consequences: from time to time two individuals occupy the same position, so that the second overwrite the display of the first. But internally each one updates its position, so that when they arrive at a non-overlapping position, they are again distinct. Moreover, the Track_Zooids operation recovers the final position of each active individual, even if two or more positions overlap.

7.3. The temporal dimension

The concurrent behavior of the Tunicate individuals provides a fine control of the temporal dimension. At the individual level, each individual possesses its biological clock (Laval, 1995); at the program level, the simulation relies on a

Table 1
Objects included in the root object Modeler

Objects included	Operations provided
Parameters	Display_Config_File;
Simulation_Instruments	Start_Clock;
	Stop_Simulation;
	f. Simulation_Is_Over
	f. Simulation_Is_Complete;
	{Increment...}
	Display_Results;
Space_Management	Set_Frame;
	Set_Position ();
	Request_To_Move ();
	Request_To_Fall ();
	Clear ();
	f. Next_Position ();
	f. Next_Direction ();
Computer_Interface	Clear_Screen;
	Hide_Cursor;
	Recall_Cursor;
	Pause;
	Intro_Screen;
Tunicates	Create ();
	Track_Zooids ();

f.: function; (): with parameter(s); {...} operation set.

physical clock. Individual clocks start at birth, and run independently of each other. When the simulation duration is elapsed, we need a means to stop, or at least to freeze, the individuals. According to our design principles, we forbid ourselves to act upon the individuals. They should detect by themselves that a stunning external event happened, preventing them from continuing anything. In the CALIFE program, this is accomplished by making each individual frequently "poll" the external clock object, checking out if the simulation is finished. This is analogous to sensing the environment. In a more sophisticated artificial life program, it would be conceivable that each individual be provided with a kind of "nervous system" (implemented by an Ada task nested within the general behavioral task of each individual).

7.4. Further levels of hierarchical decomposition

The five child objects issued from the root object decomposition are shown in Fig. 1. The Table 1 shows their provided operations. The Simulate_Bloom object in Fig. 1 is an OPCS controlling the actions of the child objects. The Parameters object is a terminal object. Reading a configuration file is part of its initialization, so that this operation is not shown in Table 1.

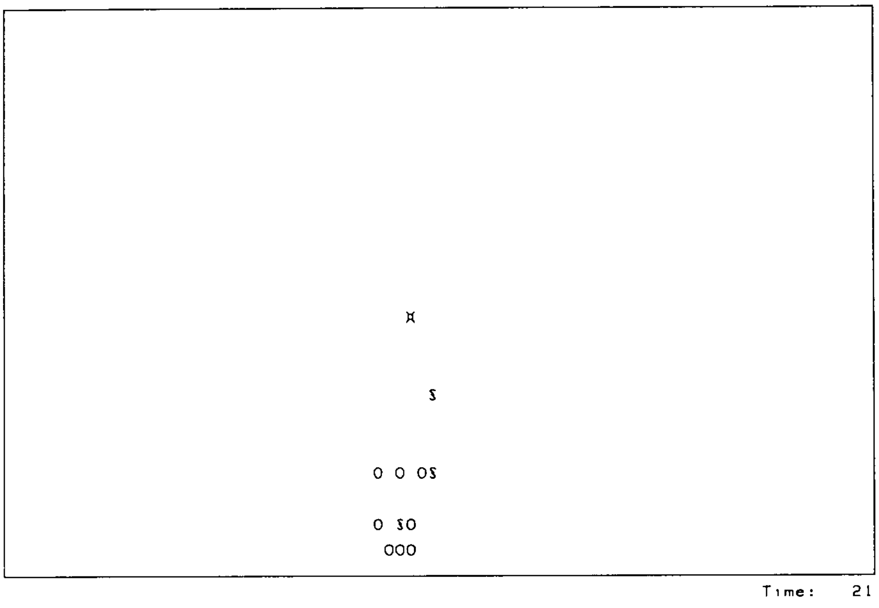
The configuration file is an ordinary ASCII file. This allows change in the initial parameters values with a simple text editor, without recompiling the program. The Parameters object provides most of the values, or initial values, of the variables used in the simulation, like the simulation duration, the number of blastozooids per chain, etc.

The Simulation_Instruments object provides two kinds of services, implemented by two child objects, Simulation_Clock and Zooid_Counter. The first of these objects gives access to the services of the simulation clock. The operation Start_Clock is used by Modeler at the beginning of the simulation. The clock has visibility over the Simulation_Duration parameter, so that it stops when this amount of time is elapsed. Each Tunicate inquires the selector Simulation_Is_Over and stops immediately with the clock. If the Tunicate bloom has consumed all the available computer memory by reproducing without restraint, the first Tunicate experiencing lack of material calls Stop_Simulation, which sets Simulation_Is_Over to True. The selector Simulation_Is_Complete returns True when the simulation is finished and there are no more reproducing Tunicates, so that Modeler can safely use the Track_Zooids operation. The other operations of Simulation_Instruments are implemented in the child object Zooid_Counter. The operation marked {Increment...} is an operation set (a writing facility standing for several akin operations). These operations increment different counters for the zooid categories (new oozooids created, new chains, oozooid cadavers, etc.). The Display_Results operations reads the counters, and also the clock status.

The operations implemented by the Computer_Interface object (Table 1) are accomplished by several "utility" objects drawn from a pre-existing library, except for the Intro_Screen operation. Several utility packages are taken from Jones (1989). The Intro_Screen procedure is an application-dependent routine displaying an introductory screen with the program name and version number.

In the Tunicate object, the Create operation spawns the first oozooid at a place, and with an initial swimming direction, defined in the configuration file. The Track_Zooids operation follows all zooid pointers, starting from the first oozooid created, incrementing in passing the corresponding counters. The Tunicates object is a terminal passive object, but is rather particular. It has a self-reproducing Zooid_Type component (in two alternating incarnations, an oozooid generation and a chain generation). In HOOD, the closest representation of a set of objects is a class (implemented by a generic package). The Tunicates object does not need to be generic (it has no parameters); it recursively produces active instances of its Zooid_Type pattern. Each instance of the Zooid_Type possesses an internal Ada task, responsible for the zooid behavior. This behavior is presently limited to ageing (and dying), moving, and reproducing (Laval, 1995). Feeding, respirating, excreting, and growing, which are op-

(A)



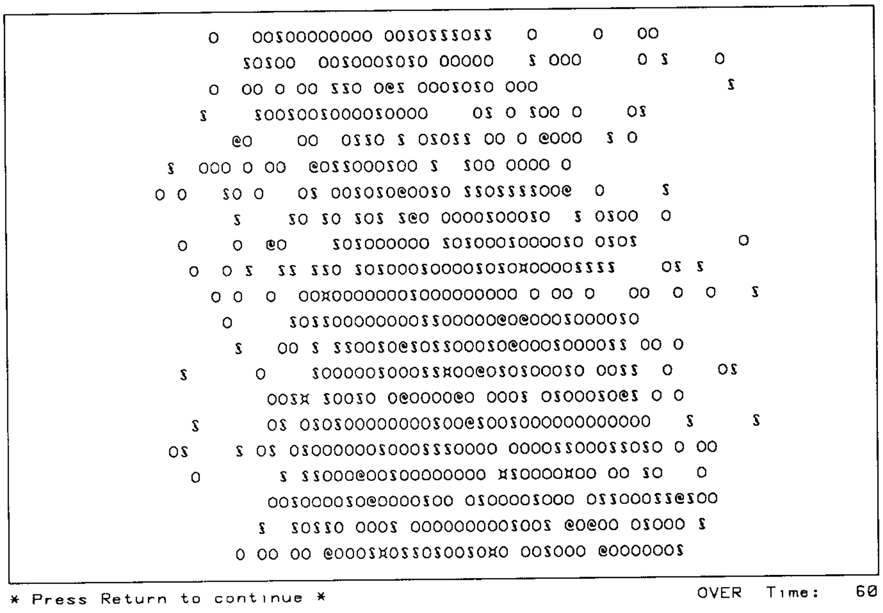
```
                          ¤


                             §


                         O O O§

                         O §O
                          OOO
                                                         Time:    21
```

(B)



```
           O    OO§OOOOOOOOO OO§O§§§O§§   O      O    OO
                §O§OO   OO§OOO§O§O OOOOO    § OOO      O §    O
           O  OO O OO §§O O@§ OOO§O§O OOO                     §
          §     §OO§OO§OOOO§OOOO     O§ O §OO O     O§
              @O     OO   O§§O § O§O§§ OO O @OOO   § O
         §  OOO O OO   @O§§OOO§OO §  §OO OOOO O
        O O    §O O    O§ OO§O§O@OO§O §§O§§§§OO@  O      §
                §     §O §O §O§ §@O OOOO§OOO§O  § O§OO   O
          O     O  @O      §O§OOOOOO §O§OOO§OOOO§O O§O§          O
           O   O §  §§ §§O §O§OOO§OOOO§O§O¤OOOO§§§§     O§ §
             O O   O  OO¤OOOOOOOO§OOOOOOOOO O OO O    OO   O   O    §
              O      §O§§OOOOOOOOO§§OOOOO@O@OOO§OOOO§O
                §    OO §  §§OO§O@§O§§OOO§O@OOO§OOOO§§ OO O
         §      O     §OOOOO§OOO§§¤OO@O§O§OOO§O  OO§§  O     O§
                    OO§¤ §OO§O O@OOOO@O OOO§ O§OOO§O@§ O O
          §         O§ O§O§OOOOOOOOO§OO@§OO§OOOOOOOOOOOO    §      §
        O§     § O§ O§OOOOOOO§OOO§§§OOOO OOOO§§OOO§§O§O O OO
         O           § §§OOO@OO§OOOOOOOO ¤§OOOO¤OO OO §O    O
                    OO§OOOO§O@OOOO§OO O§OOOO§OOO O§§OOO§§@§OO
                  §  §O§§O OOO§ OOOOOOOOO§OO§ @O@OO O§OOO §
               O OO OO @OOO§¤O§§O§OO§O¤O OO§OOO @OOOOOO§
* Press Return to continue *                             OVER  Time:    60
```

Fig. 2. Two stages of a simulation, as they are displayed on the computer screen. The initial parameters correspond to those used in fig. 5 of Braconnot et al., 1988, except that there are only 8 blastozooids per chain. Live oozooids are represented by the character "O", oozooid cadavers by " ¤ ", live chains by "§", and chain cadavers by "@". In (A), after 21 time units (days; seconds for the simulation), the first oozooid has emitted 3 chains and is decomposing; the first chain has emitted its 8 oozooids. In (B), the time allowed to complete the simulation (60 units) is elapsed; oozooids seem to dominate because a chain (composed of 8 blastozooids) is represented by just one character. On the computer screen, oozooids and chains are displayed with contrasting colors.

erations with far-reaching consequences, will be included in a next version.

8. Output of the program

In its present state (version 3.40), the CALIFE program is principally a robust framework useful for testing the overall design. The Tunicate population increase is still unlimited. This was at first necessary in order to ascertain that the program could recover gracefully if the population growth exceeded the memory limitations. The output of the program is as follows:

– The initial parameters values are first displayed so that they may be checked before launching the simulation.

– The initial oozooid appears as an "O" character on the screen, which begins to move. When its first maturation duration is elapsed, the "O" spawns a "§", symbolizing a chain, which begins to move, in a direction differing by 1/8 from that of the oozooid. Other chains may be emitted at intervals, according to the parameter values. After a maturation time, a chain emits *n* oozooids (*n* is a parameter), because the chain symbol stands for *n* blastozooids (Fig. 2). Each oozooid and each chain are emitted after a random interval (0 to 200 milliseconds), in order to desynchronize the zooid movements on the screen (and to not submerge the task scheduler with a massive arrival of requests). When an oozooid or a chain has attained its longevity parameter, it passes to the cadaver state (the symbol changes from "O" or "§", to "¤" or "@") and moves only vertically downward. If a cadaver reaches the lower frame edge and is not totally decomposed, it symbol is cleared but the Ada task continues to exist. When a cadaver is totally decomposed, the corresponding task is "completed", but a pointer linking its structure to the one of its parent still exist, so that, starting from the pointer on the first oozooid, the Track_Zooids operation can recursively visit the final state of all created zooids. The end of the simulation occurs when either the predefined time is elapsed, or a "NOT ENOUGH MEMORY TO CONTINUE" message appears on the screen.

– Pressing a key then displays the zooid counts, together with a recall of the initial parameters values. The final status (FINISHED or OUT OF MEMORY) and the clock value are also displayed.

– The symbols and (X,Y) coordinates of all the zooids present on the screen at the end of the simulation are recorded in an ASCII file.

9. Computer implementation and performances

The CALIFE program was developed with an Alsys 386-DOS Ada compiler and environment on a 80386 (20 MHz) micro-computer. The whole program (version 3.40) represents about 3000 lines of Ada (including 1200 lines of comments), distributed in 35 compilation units. The utility libraries, representing about 2000 lines of code, are not included in this total. When more than about 250 zooids are concurrently in existence, their movements begin to be slow, and are greatly impeded when 500 are present. These figures are ×4 with a 80486DX processor at 33 MHz and a video local bus. It should be noted that in character mode (24 lines × 80 columns), there are only 1760 positions available on the screen, excluding the frame. With 6 megabytes of memory, about 1500 zooids may be active without raising Storage_Error (Laval, 1995).

10. Discussion

The apparent simplicity of the root diagram (Fig. 1) may be misleading. A number of other designs are possible, and indeed may be followed, resulting in working programs. During the long elaboration of this model, however, the parsimonious solution shown in Fig. 1 was attained only after laboriously eliminating several needlessly more complicated designs.

The object context was found very helpful when making design decisions, because the ecological problem space was closely mapped to the Ada language solution space. A question such as Should the Tunicates object be "with

Space_Management;" is answered by examining directly the corresponding ecological problem.

The HOOD method imposes very constraining conditions, in contrast with others more "object-oriented" methods, where numerous kinds of relationships between objects may be expressed. This flexibility could appear at first an advantage, but finds its limits when the project gets bigger. HOOD was not at first founded on Ada just by accident. It was often stated that Ada is not "just another language". Ada is inseparable from software engineering, and is a very powerful language. This is not to say that, in other contexts, languages like C++, Smalltalk, or Lisp, may not be more appropriate. Ada has drawbacks also for modelers; for example, it is not easy to write quick exploratory models in Ada. In fact, Ada was designed to prevent this: Ada is a language primarily directed toward large, long-lasting projects for which there are safety or mission-critical constraints. But it has such possibilities that it is slowly becoming used in others areas. Readers interested in a general presentation of the Ada language may be referred to classical text-books such as Young (1984), Cohen (1986), Booch (1987) or Barnes (1989).

In CALIFE, the movements of Tunicates are restrained by the small computer screen. An interesting alternative would be to make the Tunicates move in a virtual, unlimited space, where collisions are unlikely; this can be done, but there would be nothing to display in real-time. For the first phase of the program development, it was crucial to visualize what was happening. Moreover, the limitations of the present version lie mainly in the use of characters to represent the individuals, offering a mere 25 × 80 space. A new version using pixels of different colors instead of characters is under development. It will permit at least the representation of a 640 × 480 space on common VGA video systems, and up to 1280 × 1024 with high-end graphics boards. The VGA resolution seems already large enough to not constrain the Tunicates movements.

The role of time in the CALIFE simulation is somewhat ambiguous. On one hand, the internal clock of each zooid governs their swimming movements (one move each second), and on another hand, biological events such as the first chain maturation occur on a different time basis. Clearly, what is ecologically important is the reproduction time, but it cannot be simulated in real time, otherwise it would be necessary to wait a month for the end of a simulation run. To the extent that all biological events can be proportionally scaled down, there is no hindrance to make 1 second of simulation correspond to, say, 1 day of biological time. As the Tunicate movements are not the purpose of the simulation, it is possible to state that 1 second on the screen represents, say, the average positions of 1 day movements.

This model is ultimately individual-based, because the Tunicates object makes use of an Artificial life approach. However, this is not inherent to the HOOD methodology. The limitation of the number of concurrent individuals, due to the Ada runtime scheduler, may be overcome using techniques such as the one of Rose et al., 1993.

11. Conclusion

The software design presented here should provide a robust starting basis for developing similar ecological simulations. Tunicates were chosen because their complicated life cycle presented a programming challenge; organisms with a simpler reproductive strategy should be easier to implement. Other ecosystem components may be added provided that their hierarchical level is correctly identified. With the Tunicates, a more realistic simulation would require to limit the exponential population growth. This may be done via several paths: introducing food in the spatial environment (augmenting the behavior of each Tunicate with a Eat procedure), and/or structuring the space in more or less advantageous or harmful areas, or introducing predators, parasites or parasitoids. A Eat procedure lends itself to the application of an assimilation coefficient, which regulates the age of first maturation of the offspring, the number of blastozooids per chain, and so on. The food could be phytoplanktonic or bacterial populations, with their own reproduction rate. It is easy to see that the degree of model refinement may go as far as our knowledge (or supposed knowledge) of the ecosystem. The art of the ecological modeler consists of choosing what is deemed important; the interest of the software is to allow the incorporation of this knowledge in a well integrated way, permitting one to observe the outcome. It is in the disciplined hierarchical arrangement of the objects constituting the real world complexity that a methodology such as HOOD may be helpful to the ecological modeler.

Acknowledgements

This research was supported by the Centre National de la Recherche Scientifique (CNRS/INSU, URA 716 "Écologie du plancton marin"). The author would like to thank J. Dolan and S. Dallot for their helpful comments.

References

Alldredge, A.L. and Madin, L.P., 1982. Pelagic tunicates: unique herbivores in the marine plankton. BioScience, 32: 655–663.

Andersen, V. and Nival, P., 1986. A model of the population dynamics of salps in coastal waters of the Ligurian Sea. J. Plankton Res., 8: 1091–1110.

Barnes, J.G.P., 1989. Programming in Ada (3rd ed.). Addison-Westley Publ. Co., Wokingham, UK, 494 pp.

Barnes, J., 1993. Introducing Ada 9X. ACM Ada Letters, 13: 62–132.

Batory, D. and O'Malley, S., 1992. The design and implementation of hierarchical software systems with reusable components. ACM Trans. Softw. Engin. Method., 1: 355–398.

Baveco, J.M. and Lingeman, R., 1992. An object-oriented tool for individual-oriented simulation: host–parasitoid system application. Ecol. Model., 61: 267–286.

Bailin, S.C., 1989. An object-oriented specification method for Ada. Comm. ACM, 32: 608–623.

Berner, L.D., 1967. Distributional atlas of the Thaliacea in the California current region (Tunicata). Calif. Coop. Ocean. Fish. Invest. Atlas, 8: 1–322.

Berryman, A.A., 1993. Food web connectance and feedback dominance, or does everything really depend on everything else? Oikos, 68: 183–185.

Booch, G., 1987. Software Engineering with Ada (2nd ed.). Benjamin/Cummings Publ. Co., Inc., Menlo Park, CA, 580 pp.

Booch, G., 1991. Object-Oriented Design with Applications. Benjamin/Cummings Publ. Co., Inc., Redwood City, CA, 580 pp.

Braconnot, J.C., Choe, S.-M. and Nival, P., 1988. La croissance et le développement de *Salpa fusiformis* Cuvier (Tunicate, Thaliacea). Ann. Inst. Océanogr., Paris, 64: 101–114.

Burns, A., Lister, A.M. and Wellings, A.J., 1987. A review of Ada tasking. In: G. Goos and J. Hartmanis (Editors), Lecture Notes in Computer Science 262. Springer-Verlag, Berlin, 141 pp.

Chin, R.S. and Chanson, S.T., 1991. Distributed object-based programming systems. ACM Computing Surv., 23: 91–124.

Coad, P. and Yourdon, E., 1991. Object-Oriented Analysis (2nd ed.). Yourdon Press, New York.

Cohen, N.H., 1986. Ada as a Second Language. McGraw-Hill Book Comp., 838 pp.

Colbert, E., 1989. The object-oriented software development method: a practical approach to object-oriented development. In: Proc. Tri-Ada'89 Conf. David L. Lawrence Convention Center, Pittsburg, PA, Oct. 23–26, 1989, pp. 400–415.

de Champeaux, D. and Faure, P., 1992. A comparative study of object-oriented analysis methods. J. Object-Oriented Progr., 5: 21–33.

Delatte, B., Heitz, M. and Muller, J.F., 1993. HOOD Reference Manual 3.1. Masson, Paris and Prentice Hall, London, 175 pp.

DeMarco, T., 1979. Structured Analysis and System Specification. Yourdon Press, New York.

Foxton, P., 1966. The distribution and life history of *Salpa thompsoni* Foxton with observations on a related species *Salpa gerlachei* Foxton. Discovery Rep., 34: 1–116.

Fraser, J.H., 1961. The oceanic and bathypelagic plankton of the north-east Atlantic and its possible significance to fisheries. Mar. Res. Scot., 4: 1–48.

Freitas, M.M., Moreira, A. and Guerreiro, P., 1990. Object-oriented requirement analysis in an Ada project. ACM Ada Letters, 10: 97–109.

Jackson, M., 1983. System Development. Prentice-Hall, Englewood Cliffs, NJ.

Jones, D., 1989. Ada in Action, with Practical Programming Examples. Wiley and Sons, New York, 490 pp.

Jørgensen, J., 1993. A comparison of the object-oriented features of Ada 9X and C++. In: M. Gauthier (Editor), Ada sans Frontières. Proc. 12th Ada-Europe Int. Conf., 14–18 June 1993, Paris, France. Lecture Notes in Computer Science. Springer-Verlag, Berlin, pp. 125–141.

Jørgensen, S.E., Patten, B.C. and Straškraba, M., 1992. Ecosystems emerging: toward an ecology of complex systems in a complex future. Ecol. Model., 62: 1–27.

Ladden, R.M., 1989. A survey of issues to be considered in the development of an object-oriented development methodology for Ada. ACM Ada Letters, 9: 78–89.

Lai, M., 1991. Conception Orientée Objet. Pratique de la Méthode HOOD. Dunod, Paris, 322 pp.

Laval, Ph., 1995. Implementing self-reproducing artificial organisms with Ada. ACM Ada Lett., 15: 46–53.

Lhotka, L., 1991. Object-oriented methodology in the field of aquatic ecosystem modelling. In: J. Bézivin and B. Meyer (Editors), Technology of Object-Oriented Languages and Systems, Proc. Int. Conf. TOOLS, Paris, 1991, TOOLS 4. Prentice-Hall, New York, pp. 309–317.

Liu, J., 1993. ECOLECON: An ECOLogical-ECONomic model for species conservation in complex forest landscapes. Ecol. Model., 70: 63–87.

Madhupratap, M., Devassy, V.P., Sreekumaran Nair, S.R. and Rao, T.S.S., 1980. Swarming of pelagic tunicates associated with phytoplankton bloom in the Bay of Bengal. Indian J. Mar. Sci., 9: 69–71.

Makela, M.E., Rowell, G.A., Sames IV, W.J. and Wilson, L.T., 1993. An object-oriented intra-colonial and population level model of honey bees based on behaviors of European and Africanized subspecies. Ecol. Model., 67: 259–284.

Maley, C.C. and Caswell, H., 1993. Implementing i-state configuration models for population dynamics: an object-oriented programming approach. Ecol. Model., 68: 75–89.

Ménard, F., Dallot, S. and Thomas, G., 1993. A stochastic model for ordered categorical time series. Application to planktonic abundance data. Ecol. Model., 66: 101–112.

Murray, A.G., 1990. Ada tasking as a tool for ecological modelling. ACM Ada Letters, 10: 85–90.

Myers, W., 1988. Shuttle code achieves very low error rate. IEEE Software, 5: 93–95.

Rasmussen, J., 1985. The role of hierarchical knowledge representation in decisionmaking and system management. IEEE Trans. Systems Man Cybern., SMC-15: 234–243.

Reed, D.R. and Wyant, G., 1992. How safe is C++? J. Object-oriented Progr., 15: 69–72.

Rose, K.A., Christensen, S.W. and DeAngelis, D.L., 1993. Individual-based modeling of populations with high mortality: a new method based on following a fixed number of model individuals. Ecol. Model., 68: 273–292.

Rosen, J.P., 1992. What orientation should Ada objects take? Commun. ACM, 35: 71–76.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W., 1991. Object-Oriented Modeling and Design. Prentice-Hall, Englewood Cliffs, NJ, 500 pp.

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

Seidewitz, E. and Stark, M., 1987. Towards a general object-oriented software development methodology. ACM Ada Letters, 7: 54–67.

Sequeira, R.A., Sharpe, P.J.H., Stone, N.D., El-Zik, K.M. and Makela, M.E., 1991. Object-oriented simulation: plant growth and discrete organ to organ interactions. Ecol. Model., 58: 55–89.

Sequeira, R.A., Stone, N.D., Makela, M.E., El-Zik, K.M. and Sharpe, P.J.H., 1993. Generation of mechanistic variability in a process-based object-oriented plant model. Ecol. Model., 67: 285–306.

Shlaer, S. and Mellor, S.J., 1988. Object-Oriented System Analysis: Modeling the World in Data. Yourdon Press, Prentice-Hall, Englewood Cliffs, NJ, 144 pp.

Shlaer, S. and Mellor, S.J., 1992. Object Lifecycles: Modeling the World in States. Yourdon Press, Prentice-Hall, Englewood Cliffs, NJ.

Shumate, K., 1988. Understanding Concurrency in Ada. Intertext Publications, McGraw-Hill Book Co., New York, 595 pp.

Silvert, W., 1993. Object-oriented ecosystem modelling. Ecol. Model., 68: 91–118.

Walters, N.L., 1991. An Ada object-based analysis and design approach. ACM Ada Letters, 11: 62–78.

Whitcomb, M.J. and Clark, B.N., 1989. Pragmatic definition of an object-oriented development process for Ada. In: Proc. Tri-Ada'89 Conf. David L. Lawrence Convention Center, Pittsburg, PA, Oct. 23–26, 1989, pp. 380–399.

Wirfs-Brock, R., Wilkerson, B. and Wiener, L., 1990. Designing Object-Oriented Software. Prentice-Hall, Englewood Cliffs, NJ.

Young, S.J., 1984. An Introduction to Ada (2nd ed.). Wiley and Sons, New York, 401 pp.

Yourdon, E. and Constantine, L.L., 1978. Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. Yourdon Press, New York.