Use of a multitasking operating system as a setting for the simulation of an enclosed agro-ecosystem under cognitive control

R. LACROIX, R. KOK and O.G. CLARK

Agricultural and Biosystems Engineering, Macdonald Campus of McGill University, Ste. Anne de Bellevue, QC, Canada H9X 3V9. *Received 24 October 1995; accepted 5 February 1996.*

Lacroix, R., Kok, R. and Clark, O.G. 1996. **Use of a multitasking operating system as a setting for the simulation of an enclosed agro-ecosystem under cognitive control.** Can. Agric. Eng. 38:129-137. A multitasking operating system (OS/2) was evaluated as a setting for the simulation of an enclosed agro-ecosystem under cognitive control. This setting was found to be convenient for complex simulations composed of diverse, interdependent modules. This is because it allows the concurrent execution of programs written in different languages, it offers some control over the order and manner in which these are executed, and data transfer between them is fairly easy to accomplish. The execution of the simulation modules as threads was synchronized by a management module through the use of semaphores. Most of the data exchange between modules was accomplished using shared memory segments. The simulation was of an enclosed agro-ecosystem, which in this case consisted of a greenhouse containing perceptors, effectors, plants, and soil and aerial environments. The greenhouse was subjected to weather disturbances and the internal environment was actively controlled by a system which included a cognitive component. A multitasking operating system proved to be a good setting in which to run a modular, multi-level simulation involving an artificially cognitive control system.

L'utilisation du système d'exploitation OS/2 comme plate-forme pour la simulation d'agroécosystèmes clos soumis au contrôle cognitif a été évaluée. OS/2 a été trouvé opportun pour effectuer des simulations complexes, composées de plusieurs modules différents et indépendants. La principale raison est qu'OS/2 fournit des outils pour exécuter de façon concourante des programmes écrits avec des langages différents, tout en permettant de contrôler l'ordre et la manière avec laquelle ils sont exécutés. Ces outils facilitent également le transfert de données entre les différents programmes. L'exécution des modules de simulation sous forme de processus a été synchronisée par un module de gestion à l'aide de sémaphores. Les échanges de données entre les modules étaient en majeure partie accomplis au moyen de segments de mémoire partagée. Le système simulé était un agroécosystème clos qui, dans ce cas, comprenait une serre, des percepteurs, des effecteurs, des plantes, le sol et l'environnement aérien. La serre était soumise à des perturbations météorologiques et l'environnement interne était contrôlé par un système incluant une composante cognitive. OS/2 s'est avéré une bonne plate-forme pour établir une structure de simulation modulaire, à plusieurs niveaux, et comprenant un système de contrôle artificiellement cognitif.

INTRODUCTION

Enclosed agricultural production units, such as greenhouses and phytotrons, are rapidly growing in size, number, and sophistication while their productivity is being enhanced with increasingly complex control systems. In all cases, the control systems include some kind of cognitive component, most of the cognitive capacity currently being supplied by humans. Artificial intelligence techniques make it possible, however, to reduce the need for human involvement. It is likely that in the near future there will arise a need for artificially cognitive control components to direct the functioning of enclosed agro-ecosystems so as to further increase their productivity, autonomy, stability, and robustness (Gauthier 1992; Kok and Lacroix 1993).

The resulting control systems will be complex and will incorporate many virtual components, such as cognitive mechanisms, coupled to lower-level physical devices. However, before the implementation of such systems becomes feasible, much work remains to be done on the development of the methodology used in their design and on the elaboration of design criteria and constraints. Computer simulation is an effective approach with which to help accomplish these tasks. It has relatively low resource requirements, costs little compared to physical development, and allows for rapid prototyping and testing.

The simulation of complex structures may involve components that are diverse both in their roles (e.g., external environment, ecosystem, control system, etc.) and also in the software with which they are constructed. There are several ways in which such a simulation might be implemented. A first possibility is to install each component on a separate computer. Kozai et al. (1985) adopted this approach to test greenhouse control computers by interfacing them with another computer simulating a greenhouse climate. Following this method, one might even install a single control component in a distributed fashion on a number of specialized hardware devices. A second possibility consists of installing all of the components on one computer as one large program, as has been done in traditional simulations. A third approach is to maintain the components as separate programs, but to install them all on one computer under a multitasking operating system such as UNIX or OS/2. Modularity is thus preserved, simplifying the development of individual components and allowing for the substitution and testing of different versions of the programs. This approach also permits the use of different programming languages, and even complete software packages, within the simulation. Installation of the entire set of components on a single physical

machine also has the major advantage that it is simple to establish high-speed communication channels between them. As well, less equipment is required than for the other approaches, system failures due to hardware problems are less likely to occur than with a set of cabled computers, and the entire simulation can easily be controlled from a single console.

OBJECTIVES AND SCOPE

The objective of this project was to evaluate a multitasking operating system (OS/2) as the setting for a complex, modular, multilevel simulation. Many of the concepts and terms used below were previously described by Kok and Lacroix (1993). In this case, a specific instance of an enclosed agroecosystem was modelled, namely, a greenhouse under extrinsic control. The simulation included representations of the weather, the greenhouse (including the soil and aerial environments), the crop, and two extrinsic controllers. One of the controllers possessed a rudimentary cognitive capacity.

SIMULATION STRUCTURE

The context simulated in this project comprises several aspects of the external world together with a greenhouse system, as illustrated in Fig. 1 (Kok and Lacroix 1993). The external world affects the greenhouse system via virtual and physical inputs, which in this case include current and anticipated weather conditions. The greenhouse system comprises the extrinsic controllers, including both cognitive and Pavlovian machines, and the production setting. The production setting, in turn, includes the crop, the soil and aerial environments, perceptors (sensors and pattern detectors), and effectors (final control elements and operators), as well as the greenhouse itself and its intrinsic control mechanisms.

Among various other activities, the cognitive controller uses simulation to evaluate various control strategies. Thus, there are two levels of simulation: "functional" and "subjective" (Fig. 2). The functional level is the primary simulation of the physical and virtual components. The subjective level consists of the secondary simulations carried out by the cognitive controller as part of its decision-making activities. To accommodate two levels of simulation in the same structure and to allow for the inclusion of more complex cognitive controllers in the future, the execution of the overall simulation is organized into two cycles: the "functional simulation cycle" and the "cognitive control cycle". Five linked modules are involved in the functional simulation cycle: MANAGER, WEATHER, GHOUSE, CROP, and PAVLOV. MANAGER coordinates the other four modules, causing them to be activated sequentially, in an iterative manner. The functional simulation cycle consists of all activities carried out during one iteration. The cognitive control cycle is the sum of the activities performed during an activation sequence of the cognitive controller and engages a sixth module (COGNITI).

WEATHER determines the current and anticipated meteorological conditions by interpolating values found in a look-up database. Two modules, GHOUSE and CROP, together constitute the production setting. GHOUSE performs energy balances for the covers, the aerial environment, the crop, and several soil layers. The model which it uses is largely based on the Gembloux Greenhouse Dynamic Model (GGDM2) (de Halleux 1989). CROP simulates crop growth and includes many of the functions defined in the Simple Universal CROp Simulator (SUCROS87) (Rabbinge et al. 1989). PAVLOV contains the Pavlovian controller. In this case the controller attempts to follow a given setpoint plan and makes reflex decisions to regulate the greenhouse temperature through the use of effectors. In this instance the effectors are the final control elements for the heating and ventilation.

COGNITI contains the cognitive component of the control

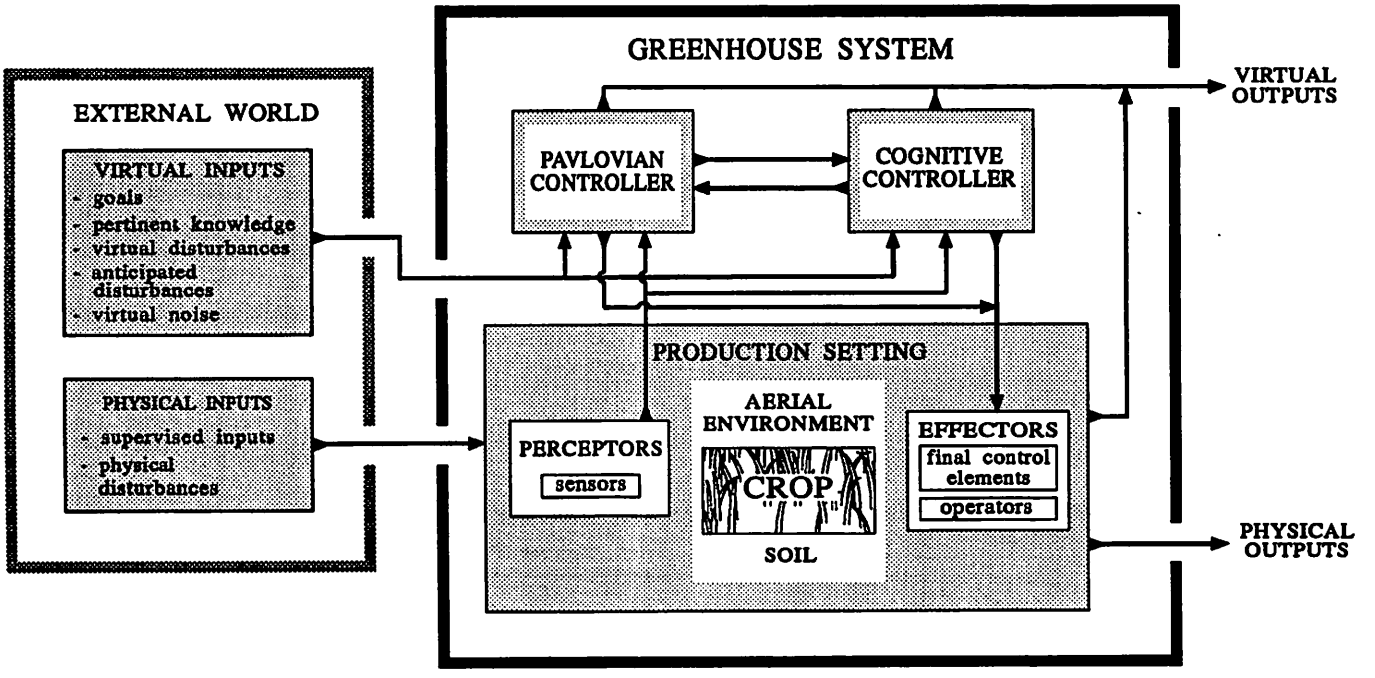

Fig. 1. Context of the simulation.

system. It has a neural model (i.e., a model consisting of a trained artificial neural network) of the entire production setting, including the intrinsic control structure. In the subjective simulations it uses the neural model to generate scenarios in response to weather forecasts and various proposed control strategies. It also has a rule-based expert system to analyze the resultant scenarios and to select the most appropriate strategy. This is then implemented by adjusting the temperature setpoint plan for the Pavlovian controller. An iteration of the cognitive control cycle (Fig. 2) consists of the activation and execution of COGNITI, any consequent subjective simulations and decision-making, as well as any consultation and adjustment of the Pavlovian setpoint plan. The interaction between Pavlovian and cognitive controllers and the use of artificial neural networks in cognitive control have been discussed in detail by Lacroix and Kok (1991), Kok and Lacroix (1993), and Lacroix (1994).

A variety of tools was used in the creation of the simulation software. The operating system OS/2, version 1.3 (IBM Corporation 1991), was used as the setting. MANAGER, PAVLOV, and WEATHER were written in Microsoft BASIC 7.0 (Microsoft Corporation 1989). Both GGDM2 and SUCROS87 were obtained as programs written in FORTRAN. The sections of interest were cut out, restructured as subroutines, and compiled with Microsoft FORTRAN 5.1 (Microsoft Corporation 1991). Using mixed-language programming features, they were then linked to main programs written in BASIC 7.0 which formed the entry ports into GHOUSE and CROP. The cognitive controller was written with GURU 3.0 (Micro Data Base Systems 1991). This is an integrated product in which an expert system shell, a relational data base, a procedural language, a text processor, and a spreadsheet are combined so that they can share data. GURU can also execute external modules. The subjective-level neural model used by the cognitive controller was constructed with NeuralWorks Professional II/Plus (NeuralWare 1991). Once the neural network was trained, it was saved as a C source file which was then compiled and linked with Microsoft C 6.0 (Microsoft Corporation 1990) as a function callable from GURU 3.0.

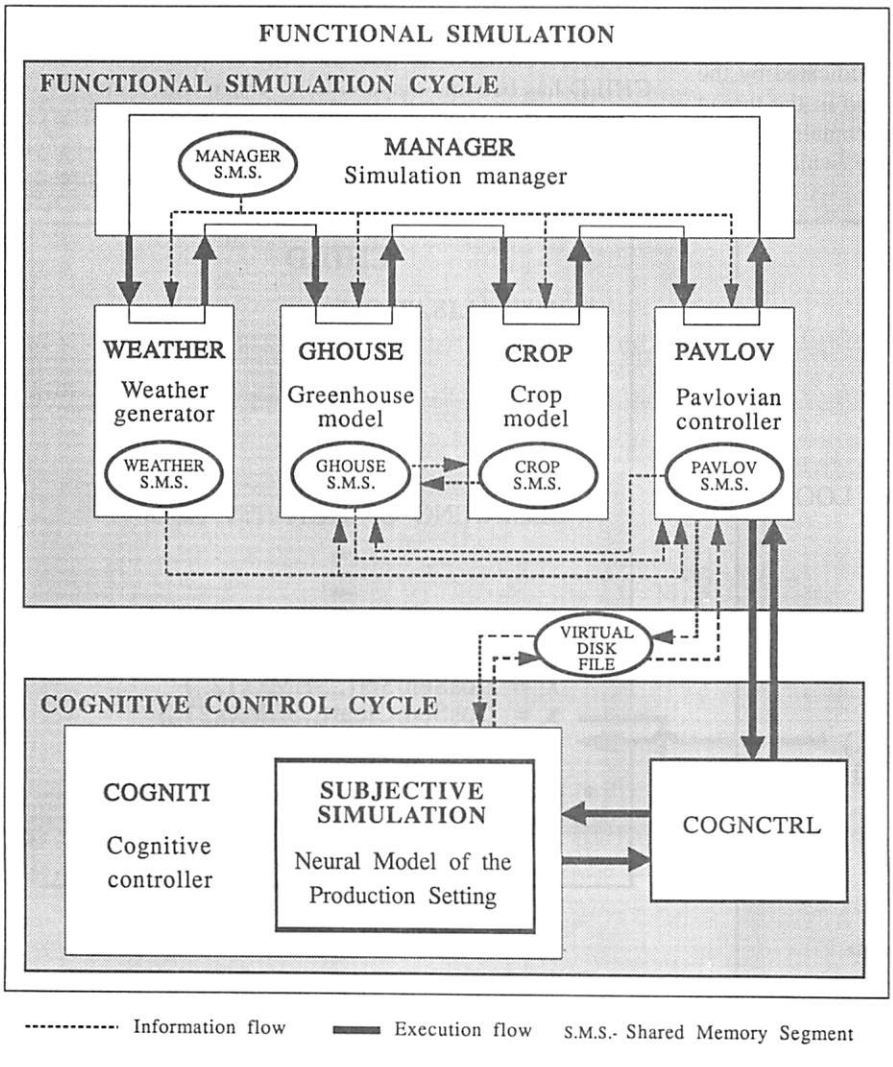

Fig. 2. Flow of execution and information in the functional simulation and cognitive control cycles.

SIMULATION EXECUTION

Utilization of OS/2

OS/2 is a multitasking operating system which groups commands according to several levels of organization and which can dynamically allocate resources such as memory, access to input and output devices, and CPU time slices. The organizational groups used in OS/2 are called "threads", "processes", and "sessions" ("screen groups"). Processes are the owners of system resources such as memory address space, files, and device monitors. In this research, the instructions composing each simulation module constitute separate threads, each of which is grouped individually as a process, all of which in turn belong to the same session. It is therefore possible to make the simulation modules all memory-resident simultaneously, while keeping separate the resources for each. However, the modules all access the same console (i.e., mouse, keyboard, and screen), allowing their evolution to be followed concurrently on one screen in full-screen mode.

The execution condition of a memory-resident thread is indicated by its "dispatching state", which may be either "dispatched" (being executed by the CPU), "dispatchable" (waiting to be executed), or "nondispatchable" (Dror 1988). A thread may be placed

in a nondispatchable state until the occurrence of an event, such as the clearance of a semaphore. By manipulating the dispatching state of the various threads, MANAGER maintains control over the order and duration of the execution of the threads comprising the four other simulation modules. This enables it to coordinate the functional simulation cycle (Fig. 2). Thus, rather than using an allocation scheme for the CPU resource based on fixed increments of physical time, as would normally be done by the OS/2 operating system, the modules are executed sequentially and each is allowed to complete all of the calculations necessary for one functional iteration.

Activation control of COGNITI, and thus of the cognitive control cycle, was arbitrarily given to PAVLOV. Whenever activated, COGNITI either consults a previously formulated setpoint plan or performs subjective simulations, makes strategy judgment decisions, and creates a new plan.

Simulation module synchronization using semaphores

The general procedure used during the functional simulation cycle to control the execution of threads is illustrated in Fig. 3. Although the method can be used to control the execution of many threads, this example involves only two: PARENT and CHILD. Figure 3 shows a representation of the code for a program which includes a "simulation loop" indicated by the gray limits. Some of the loop code is contained in the thread named PARENT. The rest of the loop code is contained in the thread CHILD and constitutes an "activity loop" which is executed during each iteration of the simulation loop.

Control of the execution of the PARENT and CHILD threads is maintained by manipulating two semaphores, SEMxx12 and SEMxx21, with calls to OS/2 Application Program Interface services (APIs). In this example, we assume that PARENT's current dispatch state is "dispatchable", so that its execution may be initiated by OS/2. Simultaneously, CHILD is "nondispatchable"; the semaphore SEMxx12 is set and CHILD must wait for it to become cleared before it can become dispatchable. It is waiting at the instruction:

X = DosSemWait(..SEMxx12..)

After executing the simulation loop for a certain time, PARENT returns to the point where it must activate CHILD. PARENT sets the semaphore SEMxx21, clears SEMxx12, and begins its wait for SEMxx21 to be cleared again. Parent does this with the following series of calls:

X = DosSemSet(..SEMxx21..)
X = DosSemClear(..SEMxx12..)
X = DosSemWait(..SEMxx21..)

Thus, PARENT is now in a nondispatchable state and CHILD has become dispatchable. When it is eventually dispatched, CHILD completes an activity loop, sets SEMxx12, and clears SEMxx21. As a result, PARENT again becomes dispatchable. As the simulation progresses, one thread be-

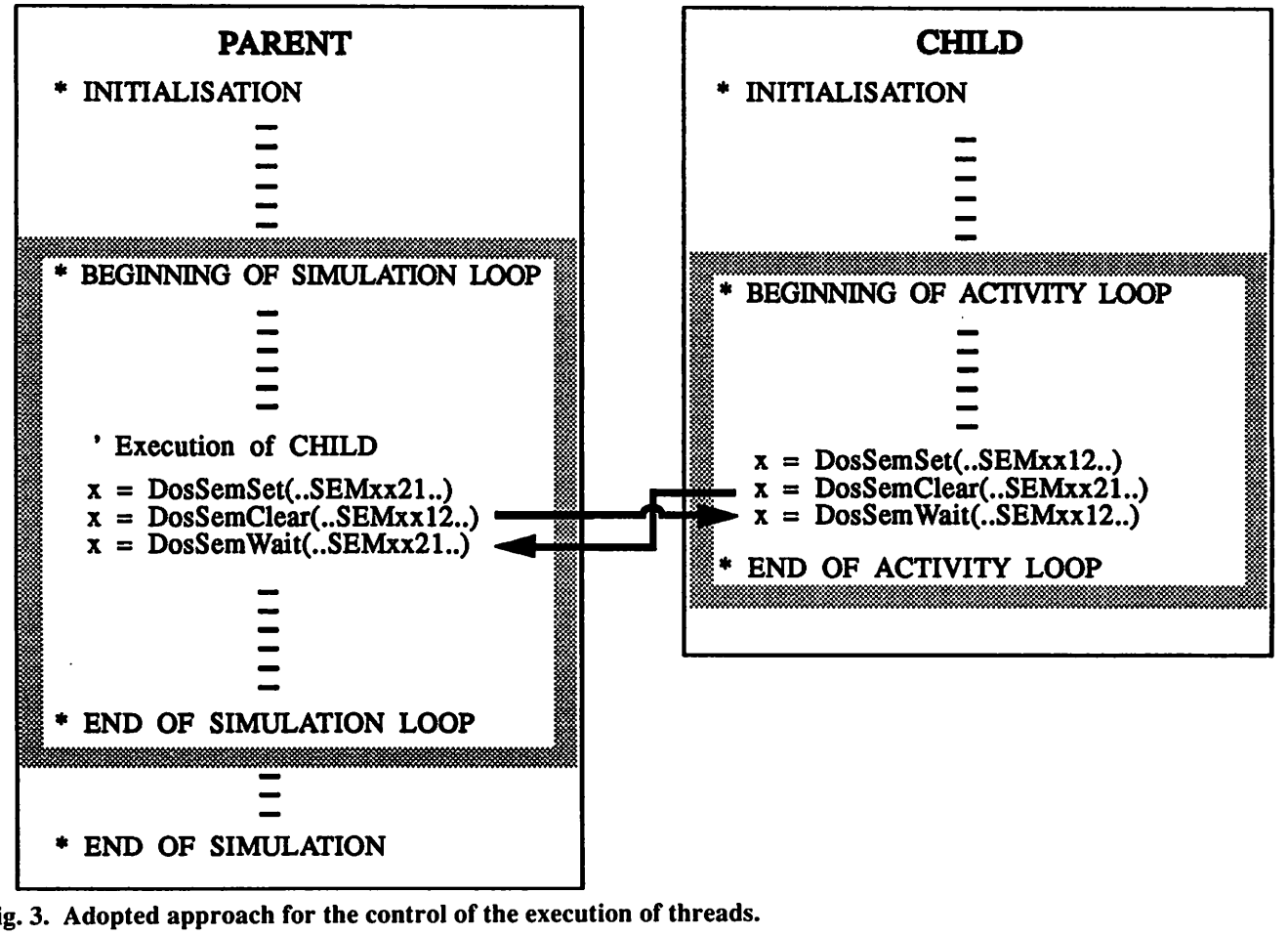

Fig. 3. Adopted approach for the control of the execution of threads.

comes dispatchable as the other becomes nondispatchable and they are thus dispatched alternately by OS/2.

The adopted procedure is safe and robust, even if, for the very short period between the times that the DosSemClear(...) and DosSemWait(...) calls are executed, both threads are in a dispatchable state. For example, (Fig. 3), CHILD might be dispatched between the DosSemClear(..SEMxx12..) and DosSemWait(..SEMxx21..) calls by PARENT. In this case, one of two scenarios may occur. CHILD might be executed in its entirety, finally clearing SEMxx21. Then, when the execution of PARENT resumes, the DosSemWait(..SEMxx21..) command in that thread will have no effect. Alternately, CHILD might be only partially executed before PARENT is dispatched. If this happens, DosSemWait(..SEMxx21..) would now have the desired effect, suspending PARENT until the further execution of CHILD is completed. A similar situation may occur during the transfer of control from CHILD to PARENT, but again, no problem would arise.

A programmer might be tempted to reduce the number of API calls employed by using DosSemSetWait(...) to simultaneously set a semaphore and begin the wait for it to be cleared, as illustrated in Fig. 4. However, between the DosSemClear(...) and DosSemSetWait(...) calls by PARENT, again, both threads will simultaneously be dispatchable. OS/2 could dispatch CHILD at this point, and no problem will arise as long as PARENT is again dispatched and sets semaphore SEMxx21 before CHILD clears it. If, however, PARENT is dispatched only after CHILD has cleared SEMxx21 then, when the command DosSemSetWait(..SEMxx21..) is issued by PARENT, a deadlock will occur and the simulation will freeze: PARENT will wait for CHILD to clear SEMxx21, and CHILD will wait for PARENT to clear SEMxx12. Therefore, for the chosen approach to work safely, both threads must use the complete command sequence: DosSemSet(..SEMxx21..), DosSemClear (..SEMxx12..), DosSemWait(..SEMxx21..).

The semaphore dependent procedure is used to synchronize the activities between MANAGER and all the threads under its control. Two semaphores are defined for each of the four modules controlled by MANAGER. Figure 5 illustrates the specifics of the adopted arrangement, using WEATHER as an example. The corresponding sections of code in GHOUSE, CROP, and PAVLOV are similar.

For the cognitive control cycle, PAVLOV keeps control with a slightly different procedure from that used in the functional simulation cycle. This is necessary because, as previously mentioned, GURU's programming language cannot be used to call APIs directly. GURU can, however, call an external program and this ability is used to manipulate the semaphores. As shown in Fig. 6, an executable file written in BASIC (COGNCTRL.EXE) and a fifth set of semaphores is employed. Whenever COGNITI finishes its activities, GURU calls COGNCTRL.EXE and then remains dormant

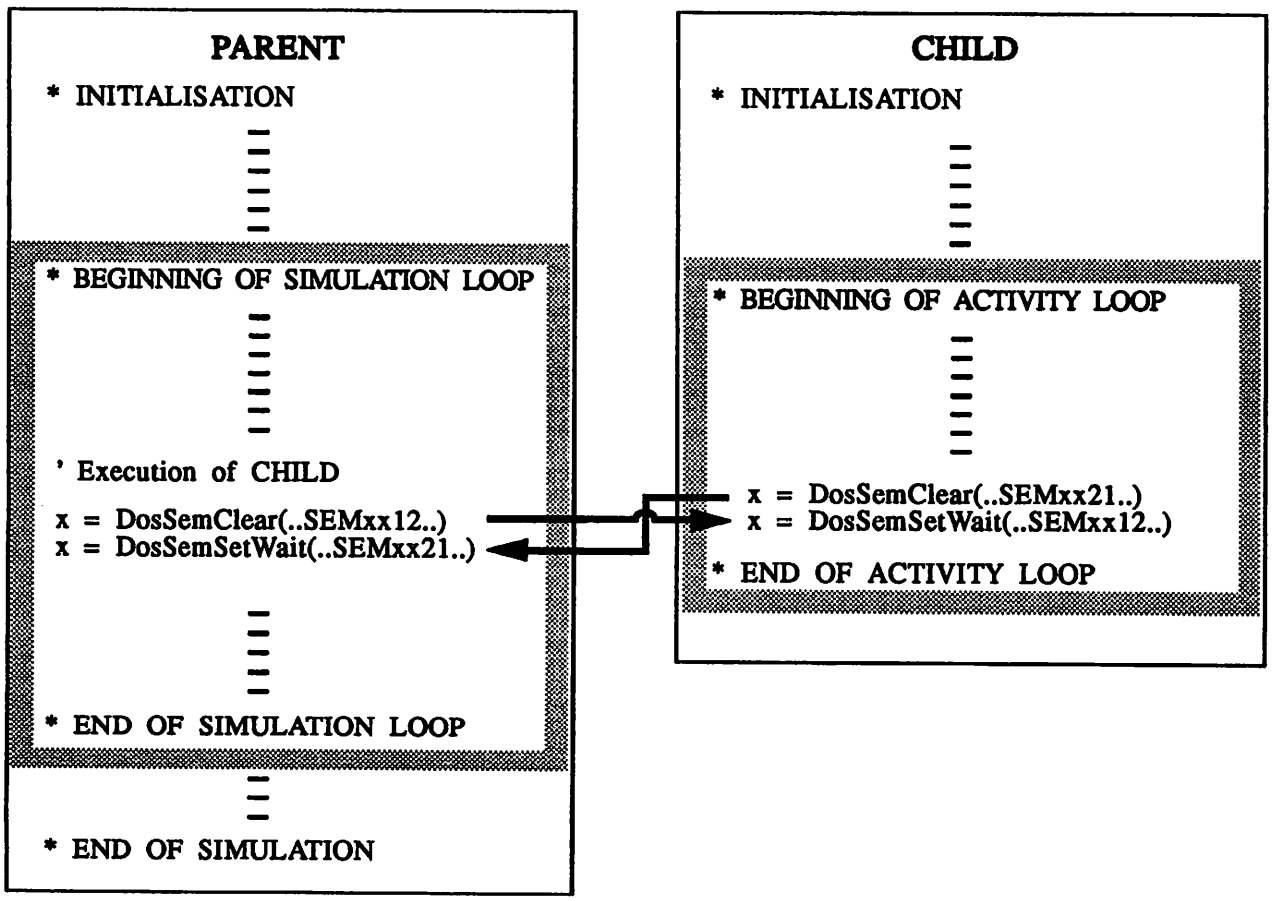

Fig. 4. The use of DosSemSetWait for the control of the execution of threads.

until that program's execution has finished. In turn, COGNCTRL.EXE sets semaphores and also becomes dormant, in effect making all of COGNITI dormant, and passing control to PAVLOV. Then, whenever PAVLOV initiates an iteration of the cognitive control cycle, it clears the semaphores, causing COGNCTRL.EXE to terminate and allowing COGNITI to proceed with its activities.

Interprocess communication

During the execution of the functional simulation cycle, the modules need to pass information to each other. This is accomplished (in the majority of cases) by using APIs to manipulate Shared Memory Segments (SMSs). The routes for interprocess communication are shown in Fig. 2. Five SMSs are used, one per module, with the exception of COGNITI. Because GURU, in which COGNITI is written, cannot directly access SMSs, communication between PAVLOV and COGNITI takes place via ASCII files on a virtual disk.

Each SMS is defined by MANAGER during the simulation initialization through a call to DosAllocShrSeg(...), at which time the size of the memory segment is specified (Fig. 5). The number of bytes allocated depends on the number of values which need to be transmitted between modules. All of the threads within the session can then access any of the SMSs. For a thread to first gain access to a SMS, it uses the API DosGetShrSeg(...) and obtains from OS/2 a segment selector that can then be used to address the SMS in subsequent operations. After that, the thread can write or read a series of bytes starting at the memory address corresponding to the segment selector.

In the functional simulation, values of both numerical and string variables are transferred from one module to another via SMSs. A simple approach is used in which numerical values are first transformed into strings, which are then written sequentially into the memory segment. The reverse procedure is used to read data from a SMS. Each string written in a SMS is preceded by a byte indicating the string length, so that when a thread needs to read a certain string in a SMS, it can determine how many bytes to read.

The management of the shared memory segments is facilitated by the fact that the sequence in which the simulation modules are executed is controlled by MANAGER. In this case, one module completes its activities and then transfers all the necessary information to the other modules, using shared memory. Thus, there is no need to tell the other modules that the sequence of information at a certain address is being changed and should not be read at that particular moment, as would be the case if the threads constituting the modules were all kept simultaneously dispatchable.

Sequence of events

At the beginning of the functional simulation, an initialization phase occurs, the first step of which is the reading of the simulation parameters by MANAGER, (e.g., beginning and end times of the simulation, report frequency). Next, the SMSs and semaphores are defined using the APIs DosAllocShrSeg(...) and DosCreateSem(...). The four threads constituting WEATHER, CROP, GHOUSE, and PAVLOV are then created by MANAGER and each passes through an initialization phase.

The sequence of API calls used for the creation of new threads is shown for WEATHER in Fig. 5. MANAGER first sets the semaphore SEMMW21 via DosSemSet(...). Using a call to DosExecPgm(...), the new thread is then created and made concurrently dispatchable with MANAGER. Next, MANAGER executes the statement DosSemWait(..SEMM21..), which is the instruction to wait for the semaphore SEMMW21 to be cleared before continuing. Concurrently, WEATHER executes its initialization phase, during which it reads its initialization parameters and gains access both to its required SMSs, via DosGetShrSeg(...), and to the required system semaphores, via DosOpenSem(...). It then sets the semaphore SEMMW12, clears SEMMW21, and begins its wait for the semaphore SEMMW12 to be cleared. Because the semaphore SEMMW21 has been cleared, MANAGER becomes dispatchable again. The same procedure is then used sequentially for GHOUSE, CROP, and PAVLOV.

During its initialization phase, PAVLOV creates COGNITI, as well as the set of semaphores used for its control (Fig. 6). When COGNITI is activated for the first time, PAVLOV sets the semaphore SEMPC21, loads and executes GURU, and waits for SEMPC21 to be cleared. On start-up, GURU accesses a procedural file containing commands specific to the current simulation. To observe the behaviour of a greenhouse when subjected to various control approaches, one of several unique start-up files is used for each simulation run. GURU then goes on to execute COGNITI's main program (i.e., the program written in the GURU command language that constitutes the main body of the cognitive controller), which performs an initialization sequence. The COGNITI main program then calls the external program COGNCTRL.EXE, which passes control back to PAVLOV through manipulation of the semaphores. This leaves GURU in memory, but suspended.

After all the modules have been created, iteration of the functional simulation cycle begins (Fig. 2). MANAGER first writes startup values into its SMS and then begins to activate the other four modules sequentially. Each carries out its activities and writes the results into its own SMS. WEATHER is activated first, using the data in MANAGER's SMS as input values. GHOUSE is activated next and reads the values in the SMSs of MANAGER, WEATHER, CROP, and PAVLOV for use in establishing the greenhouse moisture and energy balances. The last two modules to be activated are CROP and PAVLOV, respectively. PAVLOV reads values from the SMSs of MANAGER, WEATHER, and GHOUSE as inputs and every so often initiates an iteration of the cognitive control cycle by activating COGNITI. When COGNITI has completed its activities, control is again passed back to PAVLOV, thus completing the execution of that iteration of the cognitive control cycle. Next, control is given back to MANAGER to complete that iteration of the functional simulation cycle.

DISCUSSION

The ability of a multitasking operating system (OS/2) to handle many memory-resident programs at once makes possible the use of a modular, multilevel simulation structure. Modularity facilitates modification of the components, allowing experimentation with different simulation structures,

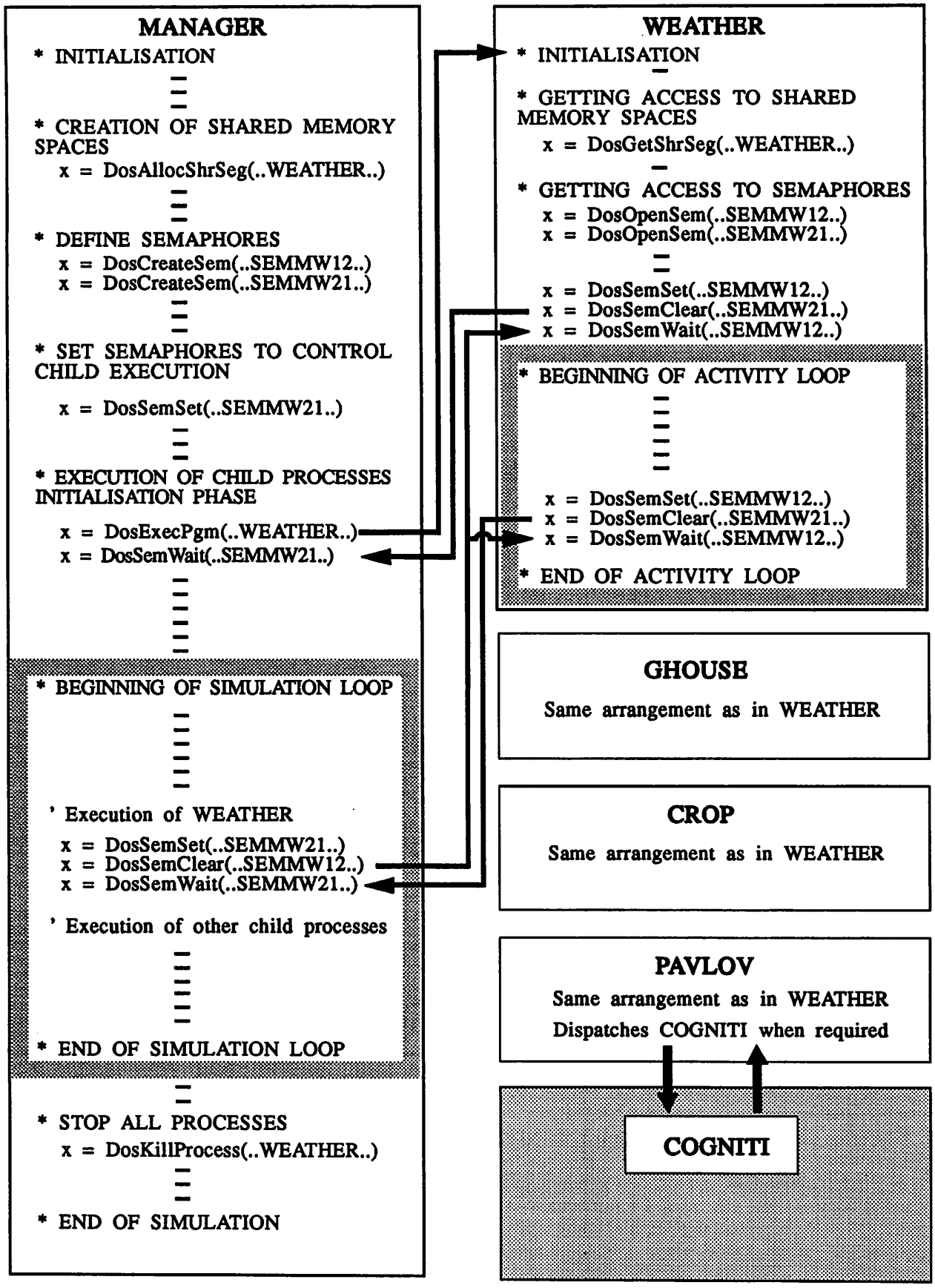

Fig. 5. Start-up and synchronization of modules in the functional simulation.

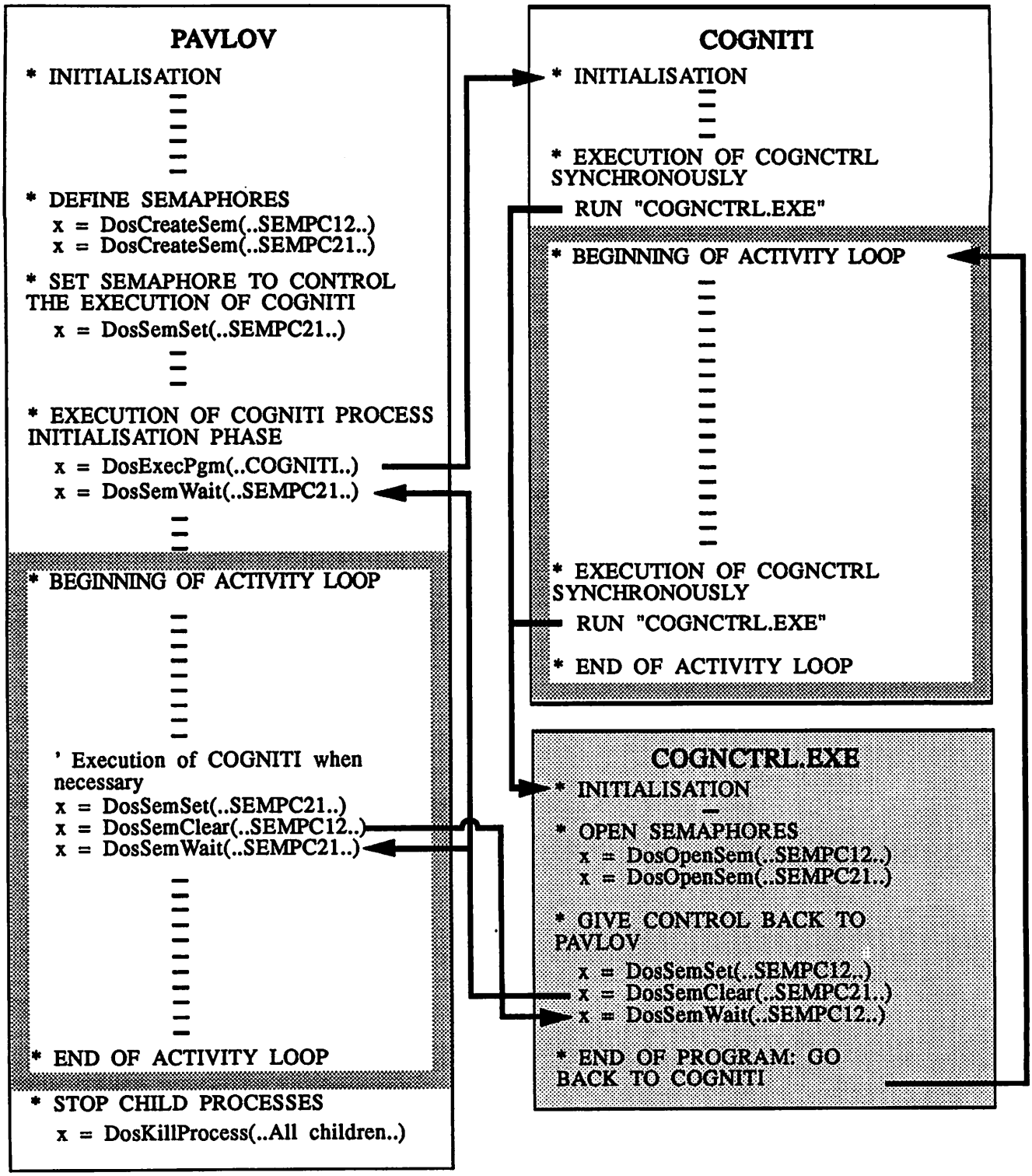

Fig. 6. Execution and control of the module COGNITI.

as well as various methods of interprocess communication and means to control OS/2 processes. Each module can be written in a different computer language and entire software packages, such as GURU, can easily be integrated into the simulation as modules. This approach was especially appropriate in this project since the simulation was of a physical system in which the components (e.g. the weather, the greenhouse, etc.) functioned independently of each other.

API services were used to accomplish data exchange between most modules via SMSs. Although this approach worked well, the procedure adopted to store numerical data in the SMSs could be made more efficient. When a numerical value is transformed into a string, as was done in this case, it requires a larger memory space. As well, the speed of data

transfer is decreased because of the conversion operations. However, due to the relatively small amount of data transferred between modules, this simple procedure was considered to be sufficient for this project. Another improvement which could be made would be to make use of functions, written in C, with which GURU would be able to pass variable values directly. Through these, COGNITI would be able to make use of SMSs, instead of having to write to an ASCII file on virtual disk, as was done in this case.

APIs were also used to manipulate the semaphores used to synchronize the simulation. Again, COGNITI could not directly access these APIs. An alternative procedure was therefore needed to synchronize its execution. The simplest method would have been to have PAVLOV load and execute GURU each time COGNITI was needed. However, this would have required a considerable amount of machine time and hard disk access. The approach adopted had the advantage of keeping in memory the values of all the variables created and reducing the number of operations performed during a simulation. In future projects, however, the control of COGNITI will be accomplished directly with MANAGER using C function calls from within GURU to manipulate semaphores.

The multitasking setting will make it relatively straightforward to write a more sophisticated control system in which the role of COGNITI will be greatly expanded beyond the derivation of setpoint plans for the Pavlovian controller. For example, other mechanisms can be added to the cognitive controller to act in parallel as independent threads, thus realizing multiple streams of cognitive activities. These activities could be made ongoing over many functional simulation cycles instead of being terminated every time the cognitive controller is suspended. Due to the modularity of the simulation structure, only minor modifications will be required to implement such a control system. Since the cognitive activities would not be run to completion at each iteration of the cognitive control cycle, a mechanism would be needed to "suspend" the execution. A mechanism would also be necessary to decide the amount of machine time allocated to each iteration. That would then depend on factors such as the desired cognitive computing ability, as well as the physical capacity of the machine on which the simulation was installed. Some of the timing, event-trapping, and priority control functions available via OS/2 APIs might be used to control the execution of the cognitive control cycle.

CONCLUSION

A multitasking operating system (OS/2) was used to implement a multi-level, multicomponent simulation structure in an innovative way. The operating system's Application Program Interface services allowed the development of an approach whereby a simulation is composed of a series of threads executed in parallel, whose dispatch is kept synchronized through the use of semaphores. The operating system's resources also allow the transfer of data between processes to be based on the use of shared memory segments. Principles similar to those developed through this research could be used with other multitasking operating systems, for example with UNIX. Also, while a greenhouse system was used in this project, the same approach might be used to develop simulators for other types of systems. Future research will be carried out to develop simulators for more diversified ecosystems.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the financial support of the Natural Science and Engineering Research Council of Canada and the Conseil des recherches en pêche et en agro-alimentaire du Québec.

REFERENCES

de Halleux, D. 1989. Modèle dynamique des échanges énergétiques des serres: Etude théorique et expérimentale. Doctoral thesis, Université de Gembloux, Belgium.

Dror, A. 1988. The Waite Group's OS/2 programmer's reference. Indianapolis, IN: Howard W. Sams and Company.

Gauthier, L. 1992. GX: a Smalltalk-based platform for greenhouse environment control. I. Modeling and managing the physical system. *Transactions of the ASAE* 35(6):2003-2009.

Kok, R. and R. Lacroix. 1993. An analytical framework for the design of autonomous, enclosed agro-ecosystems. *Agricultural Systems* 43:235-260.

Kozai, T., M.J. le Mahieu, K. Kurata and T. Takakura. 1985. A greenhouse climate simulator for testing greenhouse computers. Part 1: Operation test of ventilation control. *Acta Horticulturae* 174:413-418.

IBM Corporation. 1991. Operating System/2; Standard Edition; Version 1.3; Using advanced features. New York, NY: International Business Machines Corporation.

Lacroix, R. 1994. A framework for the design of simulation-based greenhouse control. Doctoral thesis, Agricultural Engineering, McGill University, Montréal, Canada.

Lacroix, R. and R. Kok. 1991. A cognitive controller with recourse to simulation. Paper presented at the *Conference on Automated Agriculture for the 21st Century*. St. Joseph, MI: ASAE.

Micro Data Base Systems. 1991. GURU - Integrated components manual. Lafayette, IN: Micro Data Base Systems Inc.

Microsoft Corporation. 1989. Microsoft BASIC; Programmer's guide. Redmond, WA: Microsoft Corporation.

Microsoft Corporation. 1990. Microsoft C; Advanced Programming Techniques. Redmond, WA: Microsoft Corporation.

Microsoft Corporation. 1991. Microsoft FORTRAN; Reference. Redmond, WA: Microsoft Corporation.

NeuralWare. 1991. Neural computing - NeuralWorks Professional II/PLUS and NeuralWorks Explorer. Pittsburgh, PA: NeuralWare, Inc.

Rabbinge, R., S.A. Ward and H.H. van Laar. 1989. Simulation and systems management in crop protection. Simulation Monographs; 32. Pudoc, Wageningen, The Netherlands.