An object-oriented greenhouse simulation software system - Part I: Architecture and functional description

L. GAUTHIER

Département de génie rural, Centre de Recherche en Horticulture, FSAA, Université Laval, Québec, QC, Canada G1K 7P4.
Received 20 August 1992; accepted 6 July 1993.

Gauthier, L. 1993. **An object-oriented greenhouse simulation software system - Part I: Architecture and functional description.** Can. Agric. Eng. **35**:215-222. An object-oriented software system (GX) representing a new and unique approach to the simulation of greenhouse heat and mass exchanges is described. Its main characteristic is that it greatly facilitates the study of the effect of control strategies on heat and mass balances. It thus allows for the development and validation of optimization strategies in which a dynamic behavior is desired. Its second characteristic is that it encapsulates the real time and simulation systems into a single structure. Hence, the code written to study the effect of a strategy on the greenhouse climate and energy use can be deployed and used for the real time operation of a greenhouse. Such a functionality is made possible by the use of the Smalltalk programming language which is an open-ended, object-oriented programming and runtime environment. The technology used in GX represents an alternative to conventional programming approaches, it enables the rapid development of new procedures and techniques and facilitates the transfer of knowledge to the end user. As such, it is strategic to the development of a "knowledge-based" agriculture.

Keywords: greenhouse, simulation, object-oriented programming, knowledge systems, Smalltalk

Un progiciel orienté-objet (GX) présentant une approche originale à la simulation des échanges thermiques et massiques en serre est décrit. Sa principale caractéristique est qu'il facilite l'étude des impacts d'une stratégie de contrôle sur le bilan énergétique et massique d'une serre. Cette fonctionnalité est essentielle au développement et à la validation de stratégies d'optimisation nécessitant un contrôle dynamique. Sa seconde caractéristique est qu'il encapsule un système de type "temps réel" et un autre servant à la simulation. Ainsi, le code permettant d'étudier l'impact théorique d'une stratégie sur le climat et les besoins énergétiques dans une serre peut être déployé et utilisé en temps réel pour l'opération en grandeur nature d'une serre de production. Une telle fonctionnalité est rendue possible par l'utilisation du langage Smalltalk qui est un environnement de programmation et d'exécution de programme basé sur le modèle orienté-objet et sur une architecture ouverte. La technologie utilisée dans GX représente une alternative aux langages et outils de programmation utilisés traditionnellement. Elle permet le développement accéléré des procédures et techniques et facilite le transfert des connaissances aux usagers. Aussi, cette technologie est appelée à jouer un rôle de plus en plus important dans le développement d'une agriculture "à base de connaissances".

INTRODUCTION

The simulation of agricultural systems is often problematic due 1) to the complexity and incomplete understanding of the biological processes we wish to emulate or model; and 2) the widely varying physical, technical, climatic, and topographic conditions that can be encountered. In greenhouses the problem is less severe since we can control, to a certain extent, many of the parameters that influence crop or animal growth such as temperature, humidity and light levels, gas concentrations, feeding and watering regimes, etc. For this reason, and because energy costs and climate are important in the operation of a greenhouse, many researchers have worked on the development of models to predict the heat and mass exchanges that occur within greenhouses. Such models typically take outside climatic conditions as inputs and, based on the geometry and properties of the greenhouse structure, compute heat and mass losses or gains that have occurred during a discrete time step. However, the programs implementing these models do not, generally speaking, consider the effect of dynamic control strategies on the heat balance of the greenhouse (Lacroix and Zanghi 1990; de Halleux 1989). This tends to bias the results obtained, since in modern greenhouse ranges, control strategies implemented by computers can exhibit relatively dynamic behaviors. This is especially true if the objective is to accurately predict the seasonal energy consumption of a given greenhouse structure.

For example, to maintain humidity levels within acceptable limits, a climate control program will retract thermal screens, slightly open the windows and increase the minimum pipe temperature. In other instances, when a greenhouse tends to overheat due to solar gains, the control program can be instructed to deploy the thermal screens or start the fogging system. In still other circumstances, supplemental lighting may or may not be used depending on instantaneous light levels outside, accumulated light over a 24 hour period, desired photoperiod, etc. Such measures obviously have an impact on the heat balance of the greenhouse and if the program that computes heat balances does not take them into consideration it will most likely produce results which are different from the ones that would be observed or measured. Hence, a program that can emulate the behavior of a climate control computer will not only tend to produce more accurate results, but also allow the evaluation and validation of the control strategies used by such a computer. The software described herein (GX) has such capabilities. It was designed by the author for both climate control and the simulation of greenhouse heat and mass exchanges. Hence, it not only allows the study of heat and

mass exchanges occurring in greenhouses for various climatic conditions, but also allows the study of the effect of various control strategies on these balances.

GX is in fact a greenhouse climate management shell based on the use of the object-oriented paradigm. It is designed to support a knowledge-based control of the greenhouse environment. In this article, the structure and organization of the simulation subsystem is described while the performance of a simple greenhouse model based on this architecture is described in Part II of this series of articles (Gauthier 1993c). The architecture, operation as well as other facets of the GX program are described in Gauthier and Guay (1990) and in Gauthier (1993a, 1993b).

OVERVIEW OF THE GX SOFTWARE SYSTEM

The main objective of the GX project was to design a system that enables the specification and deployment of dynamic control strategies (Challa et al. 1988). A dynamic strategy is defined as one that adjusts setpoints based on context-sensitive information such as outside climatic conditions, crop value, energy costs, weather forecasts etc. In other words, such a control strategy operates in real time and makes timely and near optimal decisions. Such supervisory control programs must obviously have continued access to data describing current conditions as well as to a means of changing the behavior of the underlying process regulation mechanisms.

The operation of GX is based on the assumption that the computing environment is distributed. In other words, that several computers are present and that they can communicate with one another. Typically, GX runs on a desktop computer and communicates with process controllers located in the greenhouses. The process controllers can be more or less autonomous or intelligent. In an environment governed by a GX application, the process controller's role is to realize setpoints and scan the input channels while the application's role is to establish setpoints. Thus, the process controller must provide the central system with information on the status of input channels (analog and digital inputs) and maintain setpoints through the comparison of desired and measured values. This corresponds to the way many modern digital control systems are structured although, in such systems, the desktop computer's responsibility is often limited to user interaction and information storage and retrieval tasks.

Since, in practice, simulations are run on a single computer, the software must be able to "emulate" the behavior of remote process controllers. The role of such an emulator is to make the use of a "virtual" environment transparent to the program that embodies the control strategy. A control strategy should be designed to operate in both the simulation and real-time environments. For the same reason, GX was designed to work with a variety of physical process controller types. Hence, the same control strategy can be used at different sites and at sites where different types of climate control computers are installed. This architecture allows the integration of the various automation subsystems found in modern nurseries (e.g., climate, irrigation, packaging, etc). It also allows the use of virtual controllers since such a process controller type can be defined.

GX tools

To achieve the functionality described above, GX was programmed using the object-oriented (OO) programming paradigm (Meyer 1988). The latter is radically different from the traditional, procedure- or algorithm-oriented one (e.g., Pascal or C) often used for the development of process control or simulation software. In general, all OO languages provide data and procedure encapsulation, object typing and subtyping, behavior and property inheritance, and polymorphism. GX was programmed in Smalltalk (Goldberg and Robson 1983) which was one of the first and most widely known commercially available OO languages. The Smalltalk programming language or environment possesses, in addition to the usual OO related characteristics, the following attributes:

- -Self documenting and inherently structured source code that can be browsed from within the runtime environment;
- -Inherent and explicit representation of logical relationships between program structures;
- -Support for incremental, modular, and open-ended software development; and
- -Support for the transparent and runtime substitution of software modules.

These capabilities derive from the use of a message passing mechanism, an incremental compiler, a multitasking kernel, and integration of both the programming and runtime environments. Interested readers are encouraged to refer to the literature (Goldberg and Robson 1983) and to Gauthier (1993a) for a more exhaustive description of Smalltalk technology. This article is limited to the presentation and discussion of the GX program structures used for simulation.

For the GX project, the Smalltalk/V PM (Digitalk, Inc., Los Angeles, CA) programming shell was used. It runs under the IBM OS/2 operating system and makes use of the OS/2 graphical user interface (the Presentation Manager). VPM supports other OS/2 features such as dynamic link libraries, dynamic data exchange, and threaded processing (Letwin 1988).

Architecture of the GX system

As described in Gauthier (1993a), the GX system contains a set of classes (i.e., data types or object categories) which can be used to model the structure and behavior of physical and virtual entities found in greenhouse complexes. In fact, most of the physical and virtual entities found in a greenhouse complex can be mapped to instances of a GX class. GX contains classes that represent sites, zones, apparatus, instrumentation units, process values, sensors, transducers, activators, crops, communications devices, etc. In addition, GX encompasses the classes that are used as templates to create instances of control strategies and the necessary logistics: protocols for information synthesis, data logging, decision making, reasoning, and calculation. The object categories used to convey and implement these protocols are described in Gauthier (1993b). However, for the benefit of the reader, their main characteristics will be presented. A running GX control strategy (or **Scenario** in GX parlance) is

carried out by various concurrent processes. One of these is in charge of periodically retrieving raw process values from the instrumentation units (which can be virtual as is the case for a simulation). These values reflect the current state of the controlled environment and are periodically logged in sequential files. A scenario can associate <<daemons>> (Winston 1984) to process values. A daemon is a fragment of code that is executed whenever the value of an object is assessed or changed. These daemons synthesize statistics such as one hour, 24 hour, or 72 hour averages, sums, maximums, and minimums. Again, the values of these statistics can be logged in sequential files or used for decision making purposes.

GX control strategies are associated with <<scripts>> which consist of decision making protocols formalized as production rules and used to convey various types of heuristic knowledge. They can be called upon periodically to assert setpoints or trigger tasks. Such tasks are typically carried out by agents which are used to generate or retrieve information that cannot be synthesized through simple heuristics. For example, a mathematical model that predicts the evolution of conditions in the controlled environment or a remote information server that supplies weather forecasts.

A GX scenario is thus a plan of action in which the various players and their respective roles and partitions are defined. When a scenario is activated, the plan is carried out. However, in contrast to the static blueprints often used in climate control programs, a GX scenario is dynamic and will establish setpoints and make other types of decisions based on context sensitive protocols.

DESCRIPTION OF SIMULATION SUBSYSTEM

General organization

The simulation subsystem that is part of GX is structured around three main components: a model used to simulate thermal and mass exchanges within a greenhouse bay, a subprogram used to simulate the evolution of climatic parameters outside the greenhouse and a process controller emulator. The organization and data flows between a control strategy and the GX simulators are shown in Fig. 1.

Since the code defining the algorithms and data structures of a simulation model reside in self-contained, encapsulated modules (classes in Smalltalk parlance), several different types of simulators can coexist within a GX system. As shown in Fig. 2., GX contains a hierarchy of simulator classes. The higher-level classes are not meant to be instantiated (i.e., they should not have direct instances). They are used to specify the minimal structure (variables) and behavior (methods) that should be implemented by the subclasses. In Fig. 2, three simulator classes are designed to be instantiated: **OutsideZoneSimulator**, **StaticHeatXchgSim**, and **DynamicHeatXchgSim**. Hence, at runtime, the user can decide which simulator should be used.

Model of operation

All scenarios share the same protocol for exchanging information with external agents. When a process spawned by a scenario wants to assess the value of a process variable such as a sensor reading or a setpoint, it sends a message to the

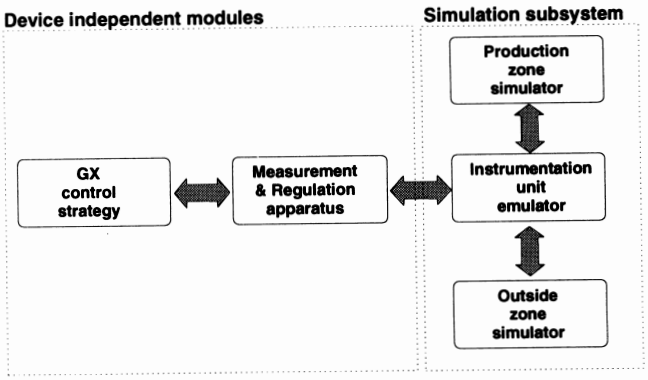

Fig. 1. Data flow in the GX simulation system. The control strategy communicates with measurement and regulation apparatus which in turn sends requests to instrumentation units. The latter obtain values from the zone simulators.

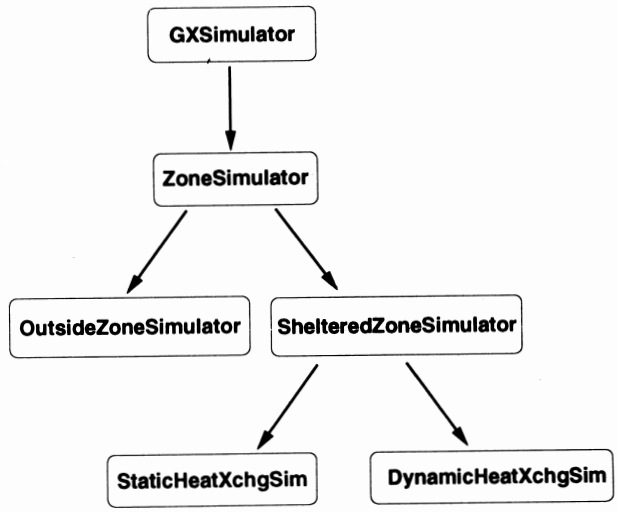

Fig. 2. Hierarchy of GX simulator classes.

appropriate apparatus requesting the value of its main variable and it receives an answer. For example (Fig. 3), to assess the temperature in a given zone, the message <<value>> is sent to the air temperature measuring apparatus associated with that zone. The apparatus receiving the message must then send the message <<value>> to the object representing the sensor. This object refers to another object representing an instrumentation unit. In Fig. 3, the instance of **AirTemp** will send the message <<getValueOf: self>> to its instrumentation unit (in this case, *self* is a variable bound to the **AirTemp** instance itself). The instance of **InstrumentationUnit** will obtain a value for the air temperature by sending a request assembled using, as needed, the attributes associated with the process value such as its address, mnemonic name, etc. In a real-time environment, the instrumentation unit sends this request to a remote computer through a physical communications link. The answer is

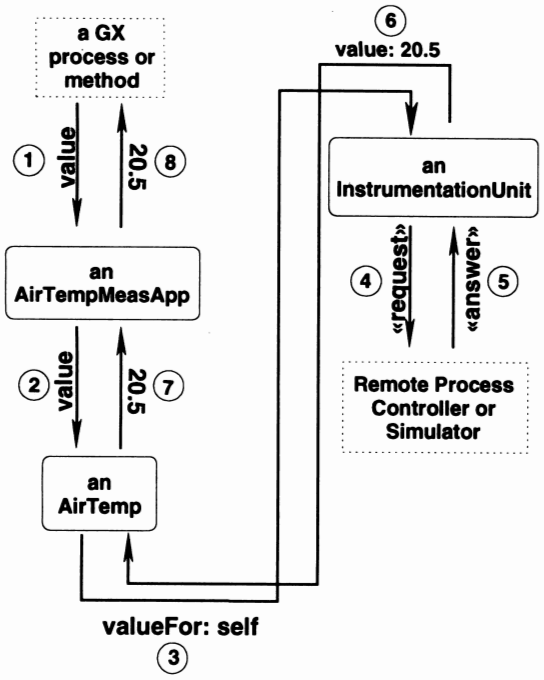

Fig. 3. Flow and sequence of messages passed between a GX process and a remote process controller.

parsed and returned to the object representing the I/O channel or sensor. In a simulated environment, the request is processed somewhat differently but, for the originator of the request, the behavior is not different. In fact, in a simulation environment, the actual value of the process variable is supplied by either a production zone or an outside zone simulator i.e., by a program running in the Smalltalk environment at the same time as the control strategy. This is possible because the Smalltalk virtual machine supports a cooperative form of multitasking that allows the simultaneous and synchronized execution of several tasks or processes (Goldberg and Robson 1983). Such a functionality is of tremendous help for program execution in both the real time and simulation modes. Priorities can be assigned to processes, and tasks can be synchronized through the use of semaphores, queues, and a real time clock.

When a GX scenario is activated, several tasks are started and run as forked or background processes in pseudo-parallel fashion. Each task is scheduled to run at a given date and time with a certain priority. Many such tasks are recurrent and thus run at fixed intervals of time (e.g., every minute, hour, day). Typically, a task to assess the value of a process variable runs every 5 minutes, one to assert setpoints every 15 minutes and tasks that log data into files can be scheduled at intervals ranging from 5 minutes to 24 hours.

Table I contains a list of tasks that are typically running during a simulation in which a single zone is active. In a real time environment, these tasks are submitted to the real time clock which interrupts the GX program every second. In a simulation environment, the tasks are submitted to a pseudo real-time clock i.e., a process running at a low priority (i.e., 2). This process advances the system clock by a user specified time step (typically 60 seconds). Hence, when all higher priority processes have terminated or are in a "waiting" state, the GX clock is advanced by one time step. In both the real-time and simulated modes of operation, the queue of scheduled tasks is continuously examined and the "ripe" events are triggered. These triggered events are run by the Smalltalk virtual machine based on their priorities.

The outside zone simulator

The role of the outside zone simulator consists simply in feeding values (i.e., readings) to the various measurement apparatus associated with the outside zone. These values correspond to the real or simulated climatic conditions that prevail outside the greenhouse. In the current implementation, the values are retrieved from a sequential file in which real or synthesized values for temperature, relative humidity, wind speed, wind direction and solar radiation are stored. Typically, a data file containing a set of values for every 15 minute interval is used. When a simulation is initiated, the user must select the file that will be used. An instance of **OutsideZoneSimulator** refers to an instance of **OutsideZone** and to the file containing weather data.

The production zone simulator

In GX, the production zone simulator is implemented by a class in which the various algorithms used to compute heat and mass balances for a greenhouse zone are specified. When an instance of **ShelteredZoneSimulator** is created, the **Pro-**

Table I: List of scheduled tasks that are active during a simulation. A higher number indicates a higher priority.

Task name	Recurrence (s)	Priority	Description
GarbageCollect	3600	7	Force a periodic garbage collect
Simulator>>tic	60	5	Advance simulation clock by one time step
OutsideZone>>updateContext	900	3	Update the context associated with the outside zone
ProdZone>>updateContext	900	3	Update the context associated with the production zone
OutsideZone>>updateDatabases	900	7	Update the databases associated with the outside zone
ProdZone>>updateDatabases	900	7	Update the databases associated with the production zone
OutsideApps>>values	300	6	Obtain values associated with the apparatus of the outside zone
ProdZoneApps>>values	300	4	Obtain the values associated with the apparatus of the production zone

ductionZone instance that it will have to simulate is specified. The simulator can thus retrieve the geometry (surface area of floor and envelope, height and slope of roof etc.) and properties of the covering material (heat transfer coefficients and transmissivity to light) from the **ProductionZone** instance it must model. Also, before the simulation process is actually started, an instrumentation unit of the type **InstrumentUnitSimulator** is associated with the simulator and with the process values attached to the zone. Thus, each time the message <<value>> is sent to a process value, the request is directed to the simulator which returns the last computed value.

In the current implementation, the following parameters are calculated in the simulation: air temperature, air relative humidity, CO_2 concentration, and PAR (photosynthetically active radiation) levels. The values of these parameters are computed every time the simulator executes the <<loop>> method which is invoked when a delay equal to or greater than the simulator time step has occurred. The <<loop>> method will recalculate the value of each parameter based on the following: the time interval, the current value of the parameter, the conditions outside the zone (temperature, absolute humidity, wind speed) and the flow rates of mass (e.g., CO_2 or H_2O injection, air infiltration, and ventilation rates) or sensible heat (e.g. heaters, solar radiation) supplied to the zone.

The request to recalculate parameters is scheduled as a background task and triggered at time intervals corresponding to the time step specified for the simulation. The structure of a **ShelteredZoneSimulator** instance is given in Fig. 4.

ShelteredZoneSimulator

Inherits from:	**GXZoneSimulator**
Inherited by:	**(none)**
Named instance variables:	
zone	Contains an instance of a zone
timeStep	Simulation time step (seconds)
timeAtLastStep	Time of last iteration
delta	Interval of time since last iteration (seconds)
params	A dictionary of zone-specific parameters used in the mathematical model
coefficients	Zone-specific coefficients used in the model
machines	Dictionary of finite state machines associated with the simulator
qNet	Net rate of heat loss (or gain) in watts
cropET	Rate of crop transpiration (L/s)
qFog	Rate of fogging (L/s)
airTemp	Current air temperature in the zone (°C)
relHum	Current relative humidity in the zone (%)
co2	Current CO_2 concentration in the zone (ppm)
par	Current amount of PAR reaching the crop (W/m^2)
Class Variables:	
(none)	
Pool dictionaries:	
HeatXchgCstes:	Dictionary of physical constants used in the equations of the model

Fig. 4. Definition of the ShelteredZoneSimulator class.

The instrumentation unit emulator

To maintain the coherence between the simulation and the real-time environments, GX contains protocols and software structures that emulate the behavior of instrumentation units (IUs). This provides considerable flexibility since the heat exchange simulation model and the IU emulator are uncoupled and can thus run separately (i.e., without requiring a mutual awareness). Moreover, the performance of the emulator can be tested through the adjustment or modification of regulation loop gains and of the underlying control logic.

The approach taken for the construction of the IU emulator is based on the definition and use of finite state machines (FSMs) and was inspired by software described by Lalonde and Pugh (1989). All finite state machines are instances of the class **FSM**. The FSMs can make use of regulation loops defined as instances of the **ProcessRegulator** class or of its subclasses (e.g., **ModifiedPIRegulator**). In Fig. 5 an example of the code used to define a FSM containing two states and used to regulate a virtual heating subsystem by virtue of a PI (proportional integral) controller is shown.

The regular and semantically clear syntax helps understand both the purpose and logic of the finite state machine. It also facilitates the tuning of the logic of the control algorithm through the addition, modification and/or removal of machine states or state lists. In the emulator currently used, FSMs to handle the heating, cooling, supplemental lighting, fogging, CO_2 enrichment, and screen operation functions were defined. Each machine is invoked, in turn, after each pass through the simulator. Thus, the boolean values of the <IF> clauses defined in the various machine states are tributary of the climatic parameters computed by the simulation model and possibly, of the states of the previously run machines. In addition, the current setpoints (also used in the <IF> clauses of machine states) are retrieved from the various apparatus which hold the setpoints for the zone being processed. The output of the machine will be a heating or ventilation demand or a change of state for CO_2 injectors, supplemental lights or fogging system valves. In turn these new values will be assessed by the simulation model and a new "system state" will be computed. In other words, the output of the FSMs is based upon the most recently computed and/or specified context and it is used as input to the simulation model.

A blackboard architecture

The control strategies, simulators, and IU emulators are loosely coupled since they communicate through the objects which represent physical or virtual entities in the zone being simulated. In other words, the set of apparatus associated with a zone acts as the interface or communications channel between control strategies and the simulation subsystem.

```
"Define a FSM for air heating "

(heaterReg :=  ModifiedPI new)      "Create a modified PI loop regulator"
      pGain: 3;                     "Proportional gain"
      iGain: 0.03;                  "Integral gain"
      lMin: 0; lMax: 100.           "Loop bounds"

"Create a Finite State Machine"
      fsm := FSM new.

"Define state #1: No heating required "
      fsm
          state: 1
                if: <condition>     "If ventilation in operation"
                action: nil         "Do nothing"
                next: 1;            "Remain in state #1"
          state: 1
                if: <condition>     "If heating demand"
                action: <action>    "Adjust heater based on PI loop output"
                next: 2;            "Goto state #2"
          state: 1                  "If no heat demand"
                orElse:  nil        "Do nothing"
                next: 1.            "Remain in state #1"

"Define state #2: Heating in operation "
      fsm
          state: 2
                if:: <condition>    "If heating demand"
                action: <action>    "Adjust heater based on PI loop output"
                next: 2;            "Remain in state #2"
          state: 2                  "If heat demand = 0"
                orElse: nil         "Do nothing"
                next: 1.            "Goto state #1"
```

Fig. 5. Smalltalk pseudo code to define a finite state machine.

In this sense, the GX simulation system can be viewed as a blackboard architecture (Nii 1986) in which the control strategies, the simulators and the emulators are the knowledge sources while the blackboard is made up of the network of objects used to represent domain entities.

DESIGNING CONTROL STRATEGIES

The design and implementation of greenhouse climate control strategies can be relatively complex. However, in GX, the polymorphism of the underlying OOPS and the relatively high level of abstraction which can be achieved with the object-oriented approach greatly facilitates the design of such control strategies. One can code relatively sophisticated control protocols with only a basic understanding of the mechanics of both the simulator and emulator components of the system.

In fact, by subclassing an existing **GXProductionScenario** subclass and by redefining a few methods out of the 60 or so methods present in the apposite superclasses one can create a functional control strategy. Generally speaking, new regulation scripts (i.e., rule sets) also have to be defined and identified in the newly created class. The concept and operation of **GXProductionScenario** subclassing was tested and verified by the author through work with graduate-level engineering students who had received about 6 hours of instruction on GX programming and operation.

USING THE SIMULATION SYSTEM

The user interface

GX was designed to serve as both a simulation platform and a real time system. To accommodate the latter, considerable efforts were devoted to the user interface. In fact, the interface allows the visualization of the behavior of the control strategy and supports user interaction. Smalltalk includes, in practically all its implementations, a window-oriented graphical user interface (GUI). Smalltalk/V PM makes use of the OS/2 Presentation Manager GUI and hence, adopts the "look and feel" of the host operating environment. The latter is essentially of the "point and click" type. It makes use of drop down menus, overlapping windows and scalable fonts and graphic objects. This, along with the multitasking capabilities that Smalltalk provides allows the coding of highly interactive and dynamic applications.

In GX, while a simulation is taking place, the user can inspect and browse through the objects and data sets used or generated by the GX control strategy. In fact, the user can even modify the control strategy by changing setpoints or by forcing a recompilation of individual methods and rule sets. In Fig. 6, a GX window used to visualize the current status (i.e., measured, desired and calculated values) of a zone is shown. The numeric gages appearing in the window are updated dynamically and hence, always reflect the last calculated or established value. When the value of a process variable changes, a message is sent to all clients of the process variable who can echo the change or take appropriate action.

Running a GX simulation

The basic program functions can be activated without the use of the keyboard. In fact, the complete system can be easily handed over to students or potential users who can experiment with its operation with a minimal amount of training and instructions.

The user must first make sure that a valid and <<runnable>> scenario is associated with the zone (or zones) that he wants to use for the simulation. A <<runnable>> scenario is one that is in a READY state. As described in Gauthier (1993b), the scenario contains or embodies the control strategy as well as the protocol for data collection and information synthesis. Hence, it is the GX component responsible for the collection of data describing system behavior (climatic parameters, energy use, etc). Once this is verified, the user starts the simulation by selecting the <RUN SIMULATION> item from the GX menu. The operator is then prompted for the name of a file containing weather data (this is done through a standard OS/2 dialog box) and for a simulation time step (the default is 60 seconds). The simulation then proceeds until it is interrupted or terminated by the user or until the end of the file containing weather data is reached.

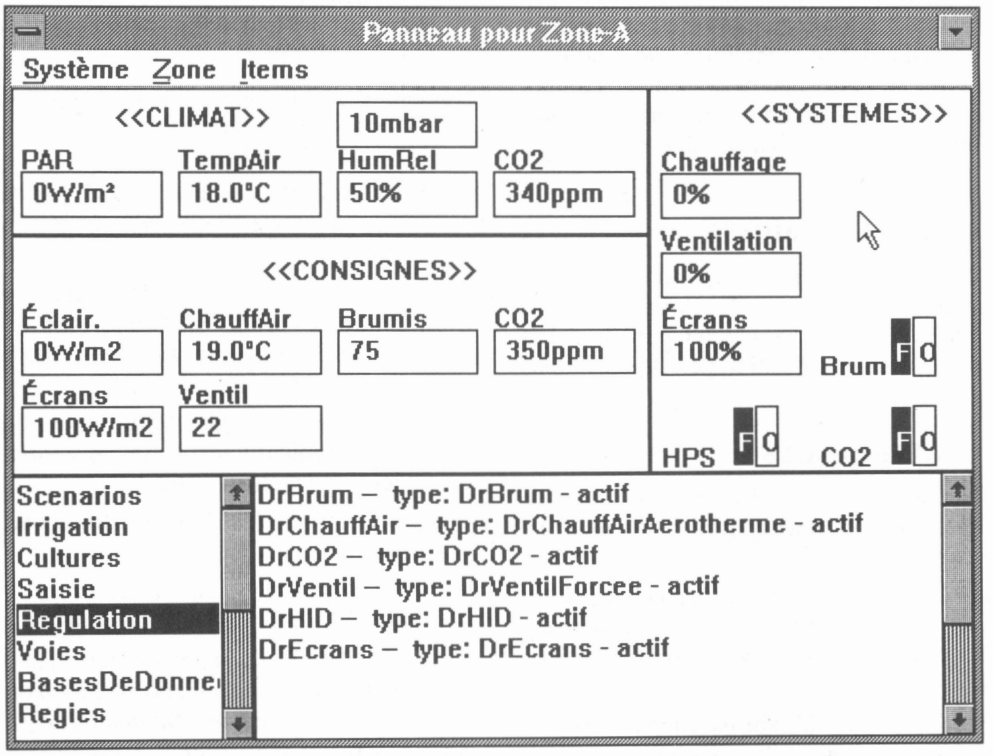

Fig. 6. Window used in GX to visualize the status of a production zone. The top left and top right panes contain current measured or calculated climatic or system parameters, the center pane contains current setpoints. The bottom right pane contains a list of object categories associated with the zone while the bottom right pane contains the list of regulation apparatus attached to the zone.

A simulation is run as a background task so the rest of the GX program remains active and usable throughout its operation. In other words, while a simulation is running, the user can examine the state of the system by opening inspector windows on zones (production or outside zones) and on the objects associated with these zones (instances of apparatus, process variables, scenarios, databases). In fact, at any time, the user can open a window on a database to look at the recent history of the simulation (trends for climatic and system-related parameters, decision logs etc). This, as well as the simulation clock which is continuously refreshed, allows the user to determine how well and how quickly the simulation is proceeding.

The dynamic and multi-tasking nature of GX also implies that changes to most of the system's components can be made while a simulation is running. For example, the user can decide to increase the heating or lighting capacity for a zone and immediately witness the effect of that change on system behavior. Changes can also be made to control loop parameters if such parameters are used by the IU emulator. Alternatively, the control strategy can be modified by changing one or several of the rule sets associated with the corresponding scenario. These rule sets are responsible for the dynamic establishment of setpoints. Hence, a change in a rule affects all the zones that use the scenario. Another way of changing the behavior of a control strategy without affecting other zones is by modifying one or several of the parameters that the strategy employs. For example, a setpoint or the price of energy can be user-specified.

Most of these changes can be specified through standard dialog boxes and a validation of the entered value is usually performed. Feedback on the effect of changes as well as on the behavior of the system is provided through the windows in which the values of most of the important zone parameters are displayed.

SUMMARY AND CONCLUSION

The software system described above and in related publications represents a new and unique approach to the simulation of greenhouse heat and mass exchanges. Its main characteristic is that it greatly facilitates the study of the effect of control strategies on heat and mass balances. This is essential for the development and validation of optimization strategies in which a dynamic behavior (i.e., one in which setpoints are determined at runtime) is desired.

Its second characteristic is that it encapsulates both the real time and simulation systems into a single structure. Hence, code written to study the effect of a strategy on greenhouse climate and energy or raw material consumption can be used for the real time operation of a greenhouse.

A third characteristic is that due to the use of the object-oriented paradigm, simulation models can be relatively easily added, replaced or enhanced without requiring changes to other parts of the overall system.

Such a functionality is made possible by the use of an open-ended, object-oriented programming and runtime environment such as Smalltalk. GX accelerates the development and study of new procedures and techniques and facilitates their transfer to the end user. For such reasons, the technologies and constructs used in GX belong to the array of tools that can favor the development of a "knowledge-based" agriculture.

ACKNOWLEDGMENTS

The author acknowledges the financial support of the Canada-Quebec agreement on agricultural development, of the National Science and Engineering Research Council, and of the Centre Francophone d'Informatisation des Organisations (CEFRIO).

REFERENCES

Challa, H., E.M. Nederhoff, G.P.A. Bot and N.J. van de Brack. 1988. Greenhouse climate control in the nineties. *Acta Horticulturae* 230:459-470.

de Halleux, D. 1989. Modèle dynamique des échanges énergétiques des serres: Étude théorique et expérimentale. Unpublished Ph.D. dissertation. University of Gembloux, Gembloux, Belgium.

Gauthier, L. 1993a. GX: A Smalltalk-based platform for greenhouse environment control. Part I - Modelling and managing the physical system. *Transactions of the ASAE* 35(6):2003-2009.

Gauthier, L. 1993b. GX: A Smalltalk-based platform for greenhouse environment control. Part II - Designing and implementing control strategies. *Transactions of the ASAE* 35(6):2011-2020.

Gauthier, L. 1933c. An object-oriented greenhouse simulation software system - Part II: Description, validation, and use of a simple model. *Canadian Agricultural Engineering* 35:223-228.

Gauthier L. and R. Guay. 1990. An object-oriented design for a greenhouse climate control system. *Transactions of the ASAE* 33:999-1004.

Goldberg A. and D. Robson. 1983. *Smalltalk 80: The language and its implementation.* Reading, MA: Addison-Wesley.

Lacroix, R and J.C. Zanghi. 1990. Étude comparative de la structure des modèles de transfert d'énergie et de masse dans les serres. *Canadian Agricultural Engineering* 32(2):269-284.

Lalonde, W. and J. Pugh. 1989. Finite state machines (automata) and constructor classes. *Journal of Object Oriented Programming* 2(6):57-62.

Letwin, G. 1988. *Inside OS/2.* Redmond, WA: Microsoft Press.

Meyer, B. 1988. *Object-oriented software construction.* New York, NY: Prentice-Hall.

Nii, H.P. 1986. Blackboard systems: The blackboard model of problem solving and the evolution of blackboard architectures. *AI Magazine*, Summer 38-53.

Winston, PH. 1984. *Artificial Intelligence.* Reading, MA: Addison-Wesley.