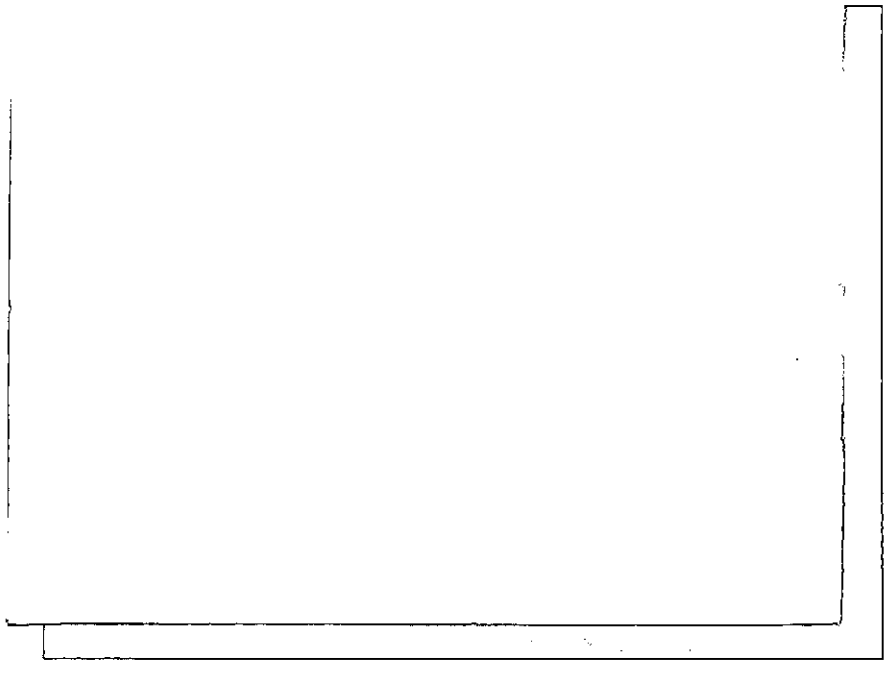

BRANZ

THE RESOURCE CENTRE FOR BUILDING EXCELLENCE

CI / SfB
(99.964)
Date August 1995

BRANZ
THE RESOURCE CENTRE FOR BUILDING EXCELLENCE

STUDY REPORT

No. 71 (1995)

OBJECT-ORIENTED GRAPHICAL USER INTERFACE FOR HOUSEKEEPING APPLICATIONS

A.H. Dechapunya
R.P. Crisostomo

The work reported here was jointly funded by the Building Research Levy, and the Foundation for Research, Science and Technology from the Public Good Science Fund

ISSN: 0113-3675

PREFACE

The Building Research Association of New Zealand (BRANZ) produced this report to document its work to date on software applications in home automation, and to suggest ways in which the home automation industry in New Zealand might benefit from further advances in this field.

ACKNOWLEDGMENTS

The work was jointly funded by the Building Research Levy and the Foundation for Research, Science and Technology from the Public Good Science Fund.

The authors also wish to acknowledge the contributions made by Dr Wayne Sharman, Dr John Duncan, and Ian Strawbridge.

READERSHIP

This report is intended for workers in software engineering, home automation, electronics and manufacturing in New Zealand.

OBJECT-ORIENTED GRAPHICAL USERS INTERFACE FOR HOUSEKEEPING APPLICATIONS

BRANZ Study Report SR 71

A H Dechapunya
R.P. Crisostomo

REFERENCE

Dechapunya, A.H. and Crisostomo, R.P. 1995. Object-Oriented Graphical Users Interface for Housekeeping Applications. Building Research Association of New Zealand, BRANZ Study Report SR 71, Judgeford, New Zealand.

KEYWORDS

Appliances, Automatic Controls, Computer Languages, Controls, Home Automation, Interactivity, Integration, Manual Controls, Object-oriented Programming, Remote Controls, Security Systems, Sensors, Smart House, Telecommunications, Thermostats, Timers.

ABSTRACT

This report describes the design and development of an object-oriented model for the home automation domain. The model developed will allow the integration of information and technologies required in the operation of an intelligent home.

Object-oriented technology makes graphical user interface (GUI) programming much easier. The complexity of GUI codes is already encapsulated in an application framework. The result is an increase in productivity.

The design of appliance control and automation is based on the concept of treating each appliance as an object. The representation has been realised with powerful object-oriented programming and real-time interaction between the computer and the home communications network. The result is an innovative user interface allowing the users of the system to interact with appliances on the computer screen.

CONTENTS

TABLES

1.0 INTRODUCTION

1.1 Background

This report represents a research output from the study of object-oriented technology using home automation as an example of an application. The project is part of a BRANZ research programme dealing with *Integrated Information and Advanced Technology*. The research programme is investigating a number of ways in which advanced technologies can be combined to improve information and communication technologies used by knowledge industries.

In this project, software engineering and communication technologies are applied to the home automation domain, which leads to understanding of the representation, storage, and retrieval of information. See previous reports for an indepth background (Dechapunya, 1992; 1993; 1994).

1.2 Objectives

The programme sets out to identify, adapt and exploit the potential of advanced technology to enhance the control of home environments, with special focus on the disabled, by providing a structure and platform for modelling, designing and representing physical processes for home applications and services. The current project consists of the following objectives and outputs:

- applying the object-oriented models developed in 1993-1994 to housekeeping applications by extending existing classes as well as creating new classes. The principal application will be in the control and automation of lights and appliances; and

- improving the first stage of the user interface by developing an object-oriented GUI (graphical user interface) for home automation applications by re-engineering the user interface from the DOS environment to a Windows environment. The term re-engineering is used here to indicate the complete transformation of the existing user interface.

1.3 Report Design

- Section 2 outlines the methodology used in the investigation.

- Section 3 describes the main classes for appliance control and automation operations and their C++ implementation, and discusses issues which arose in the investigation.

- Section 4 concludes the report.

2.0 METHODOLOGY

2.1 Moving from a Menu-based User Interface to a Graphics-based User Interface

The previous design of the user interface was menu-oriented and implemented in a DOS environment. The basic problem with the menu-based user interface is that the users are not in control of the applications. This is because a menu-based system only allows users to access the applications linearly. The interactivity between users and the applications is limited.

A better approach is to access the applications, not from top to bottom, but by any path which users choose (non-linear access). Users have their own specific ways of using the applications. They should be able to go to any option, jump to any other option and get answers any time they require answers.

Thus, there is a need to move from a DOS, menu, to a Windows, graphical user interface.

DOS (Disk Operating System) is by far the most popular operating system for personal computers (PCs). It, however, does not fit into a new environment of computing because:

- It is a single tasking system. Users need to use a number of applications at the same time.
- Different DOS applications have different user interfaces. There is no standard user interface for DOS applications.

Microsoft Windows, a Windows graphical environment, has been designed to overcome these DOS limitations. It is a multi-tasking system. It provides a consistent graphical user interface (GUI) allowing users to easily control the applications. Windows applications share common features in a user interface. Once users know how to use one application, they will find it very simple to use other Windows-based applications. Conventional applications lead the user through the flow of defined events. Event-driven applications allow the user to drive the programs through random events. Windows applications are event-driven programs (Box, 1994; Comaford, 1994; Leinfuss, 1994; Linthicum, 1994; Rimmer, 1994; Roetzheim; Tetewsky, 1994; Walrath and Hayden, 1994).

Windows has been gaining popularity at a very rapid rate. Most of the new tools and applications are implemented in a Windows environment. It was reported (Tools For Windows, January 1994) that since the introduction of Windows 3.1 in July 1992:

- More than 30 million copies of Windows 3.1 have been sold.
- Over 100,000 kits for software developers have been sold.
- Companies have been established to provide Windows-based software products.

2.2 Object-Oriented Technologies

The object-oriented technologies used in this work include object-oriented analysis and design, and object-oriented programming. An object-oriented analysis and design is a set of guidelines and frameworks to allow software engineers to develop an application model. The analysis and design of the problem domain is based on Booch's methodology (Dechapunya, 1993). C++ is used as an object-oriented (OO) programming language (Dechapunya, 1993; Mitchell et. al., 1992; Perry, 1993).

3.0 APPLIANCE CONTROL AND AUTOMATION

This chapter begins with an overview of the concept of appliance control and automation. It follows with a section on the design and modelling of classes for handling events and their respective actions. User interfaces are described in Section 3.3. Data requirements for the program are described in Section 3.4. The essential classes used for appliance control and appliance automation are described in Sections 3.5. The chapter closes with a discussion section.

3.1 Overview

The infrastructure of a Total Home Controller (THC) for controlling and automating homes is shown in Table 1. The basic system consists of:

- A standard PC running Windows 3.1 or higher.
- A Device driver for the PC interface
- A PC interface connecting the PC to the home communications network.
- Appliance interfaces connecting appliances to the home communications network.
- Application software functioning as the controller and the automator.

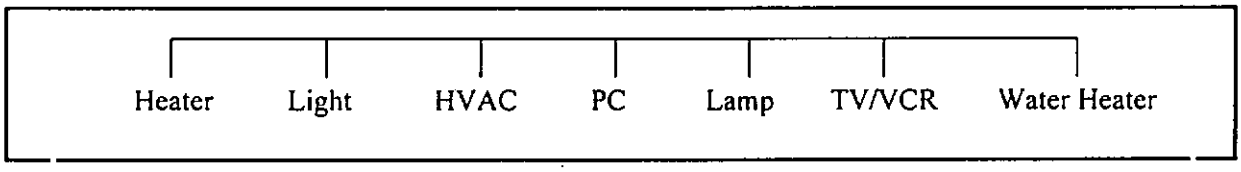

Table 1: Home Communications Network (Dechapunya, 1994)

The function of the THC is to control and automate a home. The main tasks and units which are used to perform the appliance control and appliance automation are shown in Table 2.

Tasks	Units
General management of THC	Main Unit
Control of appliances and lights	ApplianceControl Unit
Automation of appliances and lights	Automatic Unit

Table 2: Units for Modelling Home Control and Automation

The present version of THC is capable of both appliance control and appliance automation. Appliance control is the operation of appliances when they are subjected to interactive control by users. User interfaces used in this study include remotes, keyboards, and mouses.

Appliance automation concerns the operation of appliances when they are subjected to event structures. An event structure is essential in home automation applications. It includes:

- Scheduled events - appliances are set to operate at a specific time.
- Conditional events - a use of IF-THEN-ELSE to provide a means of programming events.
- Macro events - a single macro command will start multiple events. For example, a command "good night" might cause all lights to turn off, lower the thermostat and set a security system.

The combination of the above events makes a powerful vehicle for providing home services. For example, an occupancy sensor can be set up to switch on the light in a room if: it detects movement and the light switch is off and the homeowner is at home. It will, however, trigger an alarm if the homeowner is not at home. Another example is, when a sensor detects smoke or fire the system can be set up to sound an alarm, shutdown HVAC equipment and phone a fire service.

3.2 State Transition Design

Class Appliance has been extensively re-designed. The previous version of the Appliance class is limited in that it can only handle a single type of event (e.g. able to have one timer per appliance). The goal of the design of the new Appliance class is to overcome the limit of the single timer and allow users the flexibility to design their own automation events.

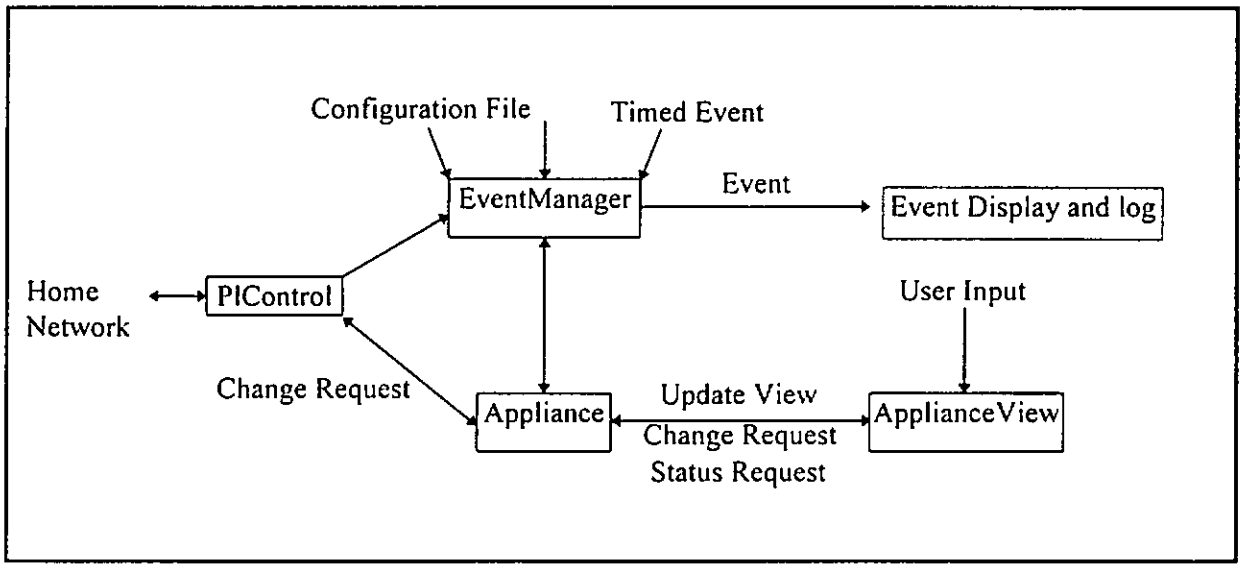

Table 3: State Transition (Event-Action) Diagram

In the new design, as shown in Table 3, a starting point is the class PowerLineController(PlControl). This class encapsulates all the function and data of the PC interface (a device which connects a PC and a home communication network). Class Appliance encapsulates all the functions and data of an appliance. This class no longer contains events of appliances. Events are now in the EventManager class. EventManager contains the data and functions required to process and manage all events. The processing of the events includes updating the status of the appropriate appliance and checking whether that event is associated with other events. The EventManager allows users to design the events to automate their homes. The events are recorded in a data file. The data is normally processed at the beginning of the system. Class ApplianceView encapsulates all the functions and data to display the properties of an appliance. This may be a display of an icon, an image, an appliance's status, etc.

To understand how these classes interact together, consider an example. A user presses a remote to turn on an outside light. The signal from the remote will be detected by PowerLineController. The control data will be stored as an event. This event will be passed to EventManager. Once received, EventManager will notify Appliance to update the status of the appliance object. Appliance will also notify ApplianceView to update all its displays. When each ApplianceView receives notification, it will request Appliance for its current status (it is not efficient for Appliance class to pass all data about an appliance to an ApplianceView object because not all data is relevant to that ApplianceView

object. In the meantime, EventManager will check the user's specific events contained in the input data file to determine whether the current event will trigger other events. Finally, the event will be recorded in an event log file. Each event will have to go through one cycle of processing.

Another example is when a user interacts with the system using a mouse or a computer keyboard. The logical path of the event processing is:

- ApplianceView receives an input;
- ApplianceView sends a request to Appliance class;
- Appliance class issues a request to PowerLineController to transmit appropriate signals;
- Appliance class issues an event to EventManager; and
- EventManager class processes the event.

3.3 User Interface

3.3.1 The Concept

The quality of the user interface is the key factor in gaining acceptance of home automation systems by a more general and non-technical consumer. The user interface must be powerful, easy to use, friendly, hide the complexity of the system, and be based on accepted standards. One of the underlying principles is to provide a uniform user interface for all applications. This means that as a new style of user interface is designed, it has to be implemented for other applications (Dechapunya, 1994).

The concept is to visualise an appliance as an object which a user can interact with. When the user clicks the appliance icon, everything about the appliance will then appear on the screen so that the user can take control of the appliance. Accordingly, the user will be able to create an appliance database containing automatic timed events (*on-off-dim-bright*) of the appliances according to: Year, Month, Date, Today, Tomorrow, Everyday, Weekdays, Weekends, Hour, Minute.

3.3.2 The Main Windows

THC's Main Windows (Table 4) have three areas: the Titlebar; the Menubar; and the General space. The Titlebar contains the name of the system which is Total Home Controller. The Menubar contains available menus, which are: Appliance; Security; Energy; Entertainment; Macro; Options; Windows; and Help. At present only the Appliance menu has been fully implemented, but the infrastructures of the other menus are in place. Clicking on the Appliance menu will result in three menu items (All Lights On, All Lights Off, and All Units Off).

Clicking on the Options menu will display two items: Configuration, and Permanent Codes. Clicking on Configuration will allow users to modify the system configuration as shown in Table 5. This dialog will ask the user to select a port which the PC interface is connected to, a House configuration file and an optional Music configuration file. This is discussed further in Section 3.4.

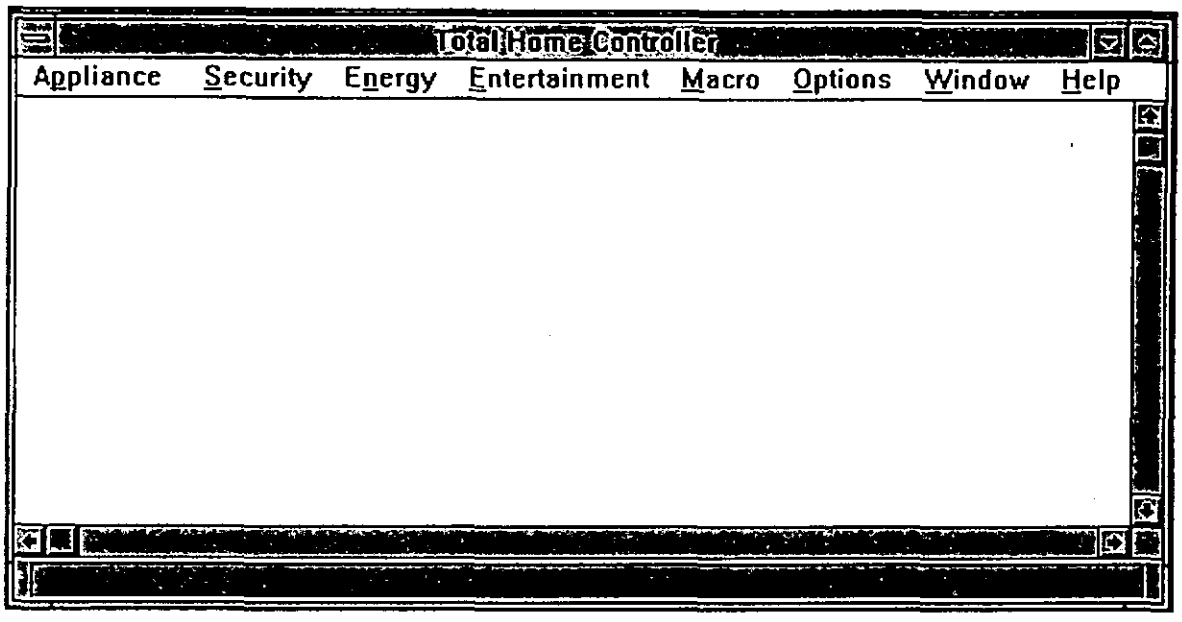

Table 4: THC's Main Windows

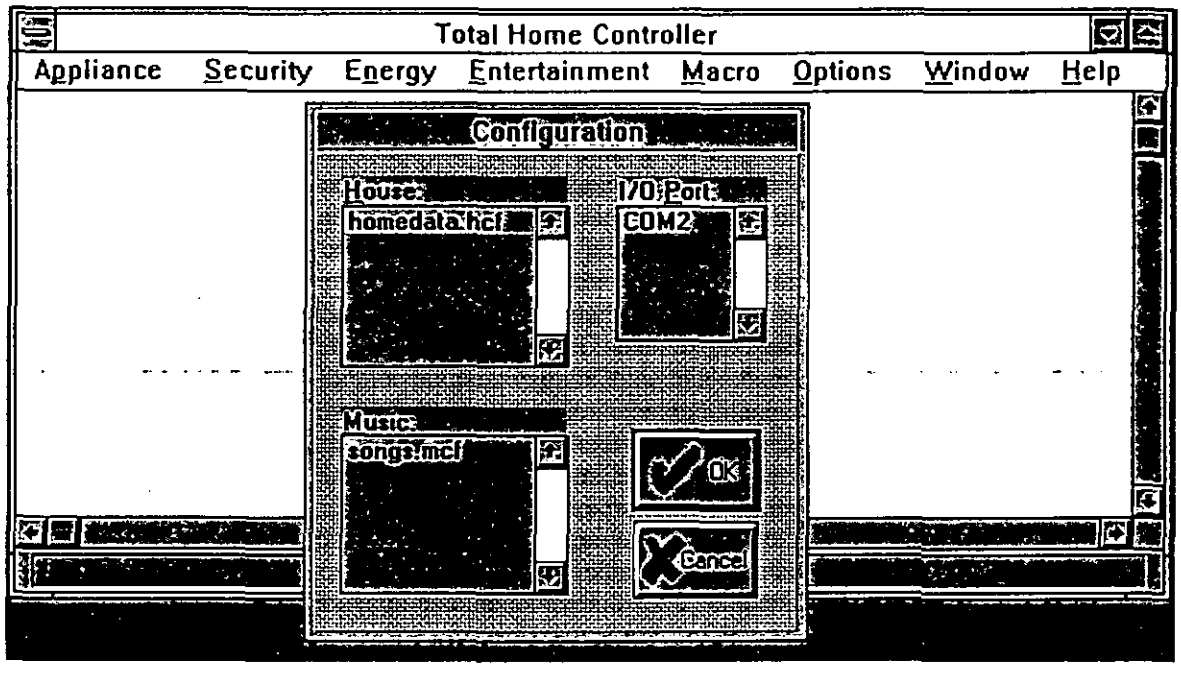

Table 5: Initialisation Dialog

3.3.3 Appliance Automation

Tables 6 and 7 show the user interface of appliance automation and the view of the appliances. Once the program starts, the Automatic routine will run continuously. THC behaves like any other Windows programs in that it can run in the background; the system is multi-tasking. The system will continuously monitor the home communications network even when a user is using other software applications on the system. The new design of the PowerlineController and EventManager classes takes advantages of this multi-tasking capability, which results in the ability of the system to handle all events occurring on every appliance.

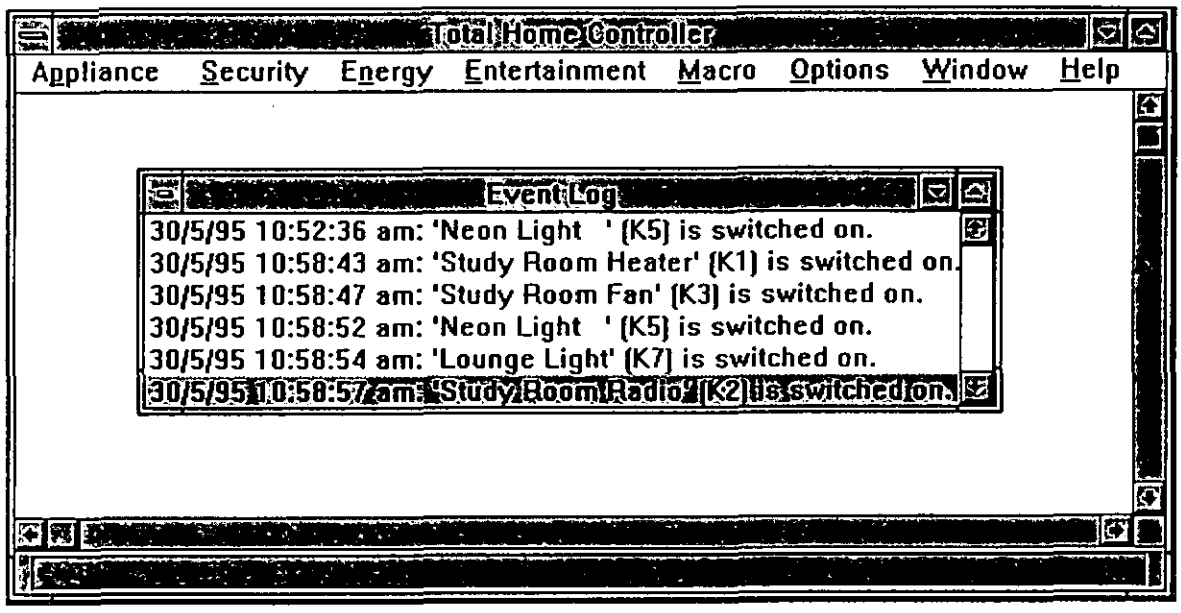

Table 6: Automation Dialog

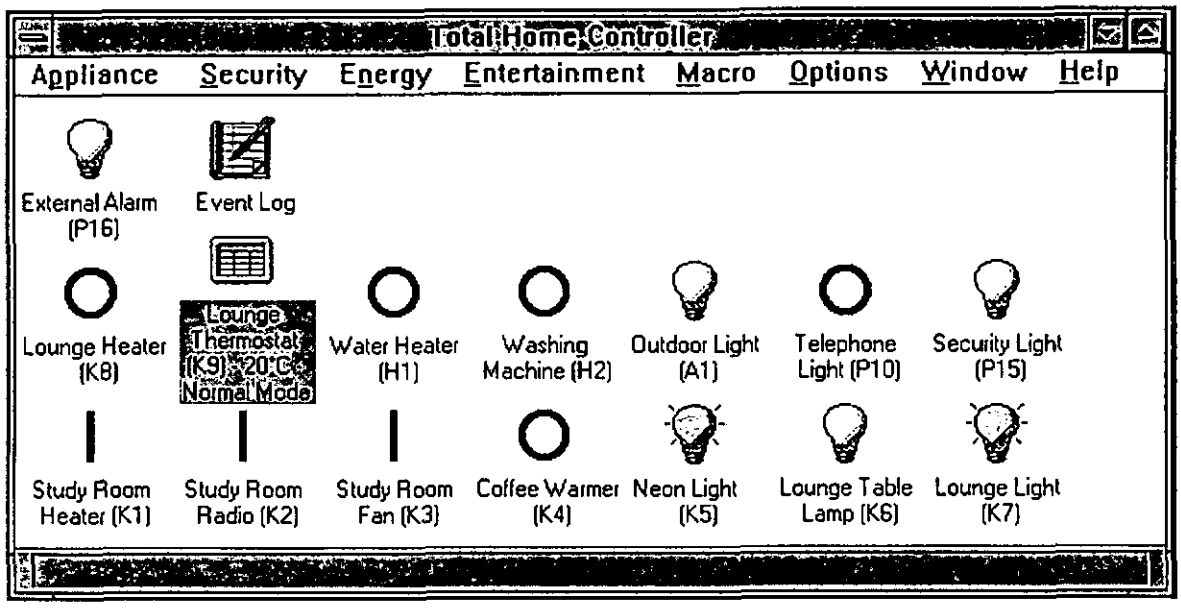

Table 7: Appliance View

3.3.4 Appliance Control

Table 8 shows an earlier design of appliance control This is basically making use of the dialog capability of Windows. As shown, the panel allows users to interactively control each appliance. The name, address, and status of each appliance are displayed. The problem with this design is that appliance control is the basis of home automation. All appliances connected to the system should be graphically displayed and accessible at anytime without having to activate any dialog.

The second design of appliance control, which overcame this problem, is shown in Table 9A. As shown, all appliances are displayed in the Main Windows. A user can visually assess the state of an appliance (on-off, light-appliance, address). A click on an appliance will bring up a menu which allows control of that appliance.

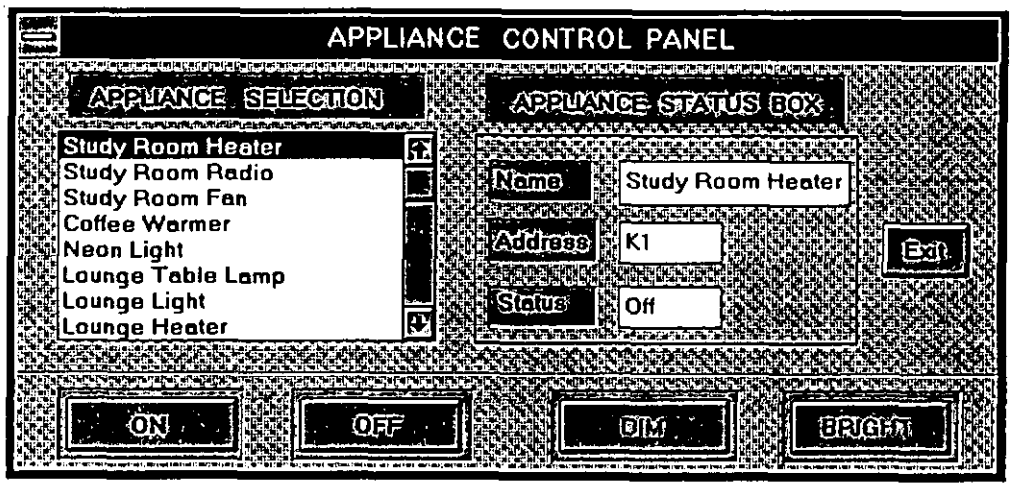

Table 8: Appliance Control Panel

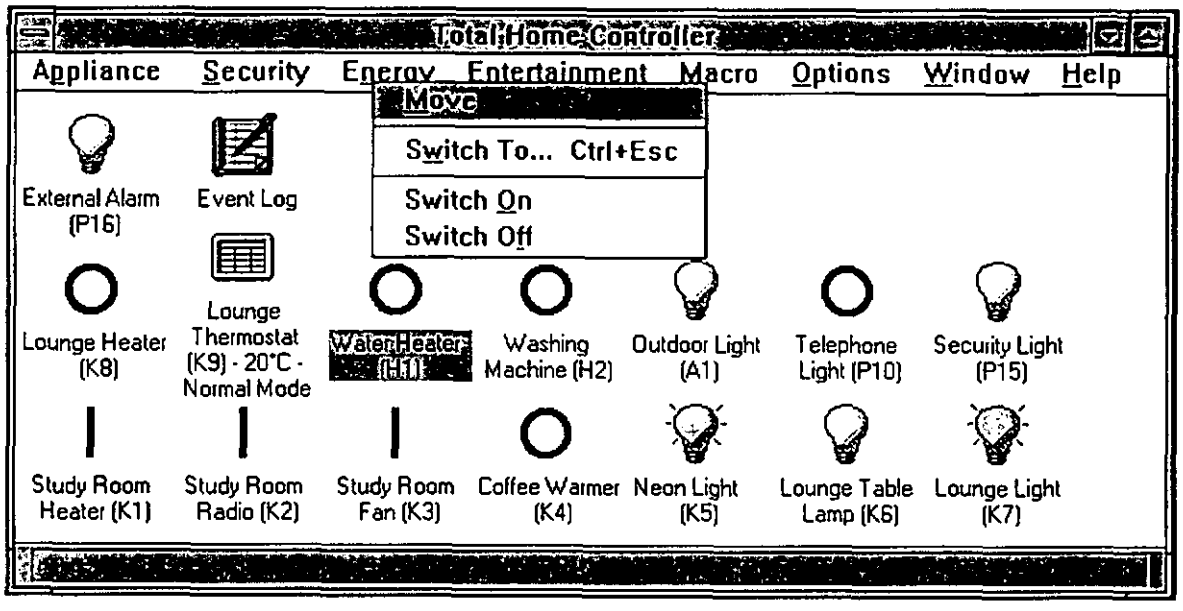

Table 9A: Appliance Control

Tables 9B and 9C illustrate the new design of appliance control for lights and thermostats respectively. The only difference between the appliance control and light control is that dim-brighten is available in the light control. The thermostat control is quite different from the appliance-light control. As shown in Table 9C, its function includes Reset, Setback mode, increment, and decrement.

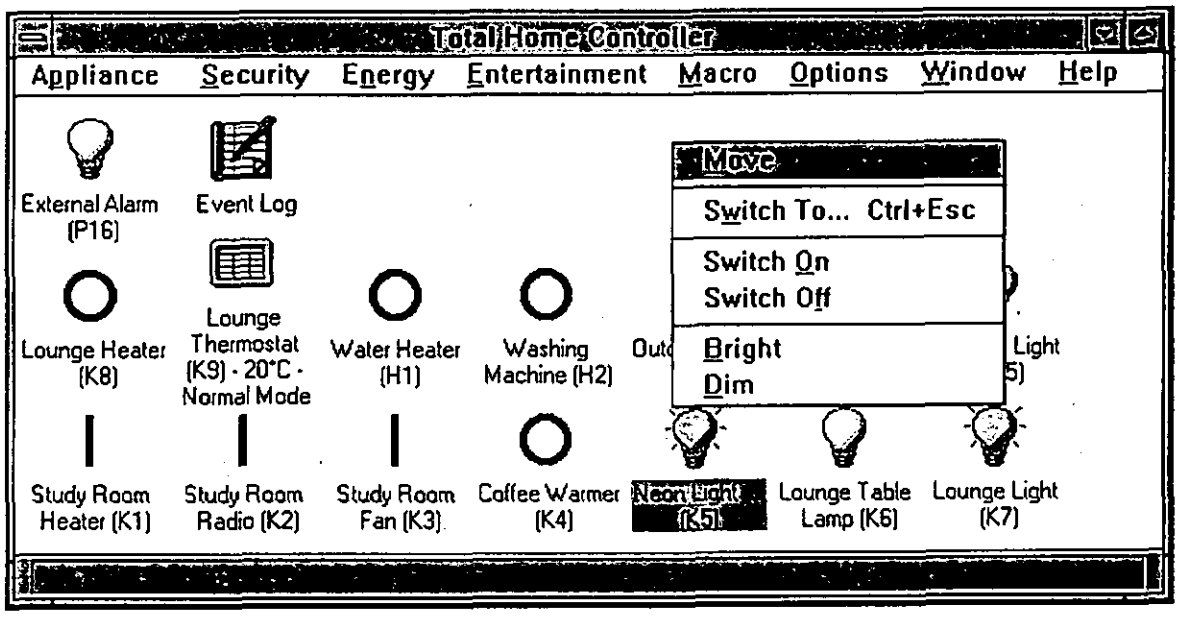

Table 9B: Appliance Control

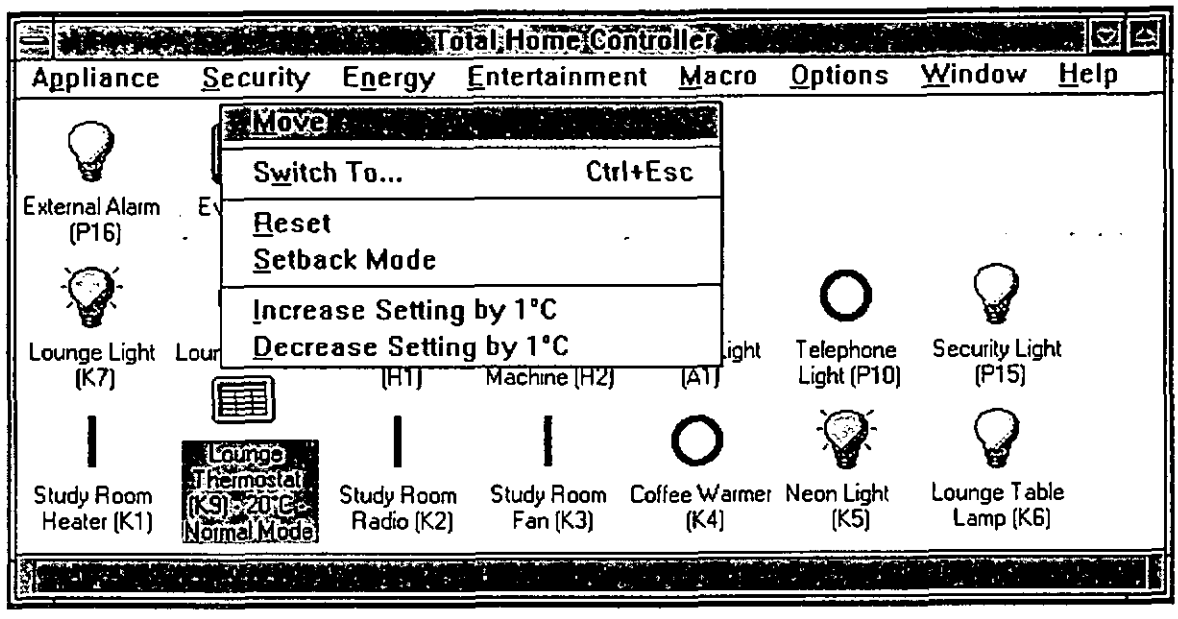

Table 9C: Appliance Control

3.4 Data Input

To be able to control and automate an appliance (appliance, light, thermostat), the appliance must have an address so that the THC can communicate and interact with it. The addressing system used here is based on the X-10 protocol (Dechapunya, 1992). In this protocol, an address consists of two symbols. The first is a letter (A to P) and the second is a number (1 to 16). A typical address is A1. The addresses of the appliances are stored in a house configuration file.

A minimum of two data files are required for the operation of the system: THC.INI and a house configuration file (e.g. HomeData.Hcf). A music configuration file (e.g. Songs.Mcf) is required if an entertainment service is needed. THC.INI is the system initialisation file. A house configuration file contains data about appliances, addresses, and their schedule of events. Examples of THC.INI and a house configuration file are shown below.

THC.INI

```
[Configuration]
Home=homedata.hcf       (a house configuration file)
Music=songs.mcf         (a music configuration file)
Port=COM1               (use serial port COM1)
[PermanentX10Codes]
PanicButton=P1
ArmAway=P2
ArmHome=P3
Thermostat=P4
MotionSensor=P5
X10SecurityHouse=P
```

HOMEDATA.HCF

```
A,    "Study Room Heater",      "K1"          (A=appliance; K1=its address)
A,    "Study Room Radio",       "K2"
T,    "K2", 10, 0, 10, 15                     (T=timer; on at 10:00; off at 10:15 daily)
A,    "Study Room Fan",         "K3"
A,    "Coffee Warmer",          "K4"
L,    "Neon Light ",            "K5"
T,    "K5", 8, 0, 16, 30,       Mon Tue Wed Thu Fri (T=timer; on at 08:00; off at 16:30 Monday
                                                      to Friday)
L,    "Lounge Table Lamp",      "K6"
L,    "Lounge Light",           "K7"
T,    "K7", 10, 0, 10, 15
A,    "Lounge Heater",          "K8"
H,    "Lounge Thermostat",      "K9", 20, 5   (H=thermostat; setpoint=20; setback=5)
T,    "K9", 22, 0, 23, 59
T,    "K9", 0, 0, 6, 0

A,    "Water Heater",           "H1"
T,    "H1", 0, 0, 4, 0
A,    "Washing Machine",        "H2"
T,    "H2", 0, 0, 4, 0

L,    "Outdoor Light",          "A1"
A,    "Telephone Light",        "P10"
L,    "Security Light",         "P15"
L,    "External Alarm",         "P16"
```

3.5 Classes For Control and Automation

The following represents the major classes used in the control and automation of appliances. Their specifications and C++ representations are described in this section.

- PowerLineController
- EventManager
- Event
- EventLog
- Appliance
- ApplianceTimer
- ApplianceView

3.5.1 Class PowerLineController

The `PowerLineController` class provides the communication link between the home network and the THC. It encapsulates the functionality of the hardware device connected to the PC. It has two basic responsibilities: to monitor the home network and capture all *events* that are *received* for later processing by other objects in the THC; and to receive *commands* from other objects in the THC and *transmit* them to the home network.

This class maintains two queues (FIFO) to accomplish these tasks: one for events received and the other for commands issued by other objects in the THC. These queues are needed because events and commands cannot be processed immediately before the next one comes along. The queues act as buffers which store the events and commands until they are ready to be processed.

Because this class encapsulates the communications link to the home network, all the other classes in the THC do not need to be concerned with the details and complexity of the communications protocol needed to receive events and transmit commands.

This class also acts as a translator between the home network and the other objects in the THC. All the data received from the home network is translated into event objects; all commands received from the other objects are translated into the data bytes expected by the home network. Therefore, it is possible to replace the hardware device with another one of similar functionality and all that needs to be modified is this class. All the other classes are shielded from this because they would still be dealing with the same events and commands.

This class uses a Windows timer event to carry out its processing. Every 50 milliseconds it checks the communications port to see if there is data which needs to be processed. If there is, the data is retrieved and processed and possibly added as an event in the event queue. If the home network is ready to receive commands, the class looks at the command queue for commands that need to be transmitted to the home network. If there is no command to be transmitted or if the home network is too busy to receive any command, this class translates the data in the event queue into event objects and puts them into the event queue of the `EventManager` class.

Table 10 describes a specification of Class PowerlineController. Table 11 shows an abstract implementation of the class using an object-oriented programming language, C++. Full listings of the source codes are not available in this report.

Class Name:	`PowerLineController`
Function:	Interface to the home network
Attributes:	`devicehandle`
	`currx`
	`RXItems`
	`TXItems`
	`eventmgr`
	`curappliance`
	`timeoutcount`
Operations:	`IsConnected`
	`Connect`
	`Disconnect`
	`On`
	`Off`
	`Bright`
	`Dim`
	`AllUnitsOff`
	`AllLightsOn`
	`AllLightsOff`
Used by:	None
Contains:	`TQueueAsDoubleList, EventManager`
Inherited by:	None
Derived from:	None

Table 10: Class Specification of PowerlineController

```
// PLCONTRL.H
// Power Line Controller class
//
class PowerLineController {
public:
   PowerLineController(EventManager *eventmgr);
   ~PowerLineController();

   inline BOOL IsConnected() const;
   BOOL Connect(const char *port);
   void Disconnect();

   void On(ApplianceAddress address);
   void Off(ApplianceAddress address);
   void Bright(ApplianceAddress address);
   void Dim(ApplianceAddress address);
   void AllUnitsOff(char housecode);
   void AllLightsOn(char housecode);
   void AllLightsOff(char housecode);
...
};
```

Table 11: C++ Representation of Class PowerlineController

3.5.2 Class EventManager

`EventManager` is responsible for managing all the events occurring in the home network by:

- Logging all events into the event logs;
- Updating the status (on/off) of all `Appliance` objects defined in the home configuration;
- Performing as an automatic timer for all appliances with `ApplianceTimers`;
- Performing security tasks;
- Performing housekeeping tasks; and
- Performing energy management tasks.

All events occurring in the home network pass through this class. This class maintains a queue which is filled by the `PowerLineController` class with `Event` objects. These `Event` objects are created by the `PowerLineController` based on the data it receives from the home network. The `EventManager` class retrieves those `Event` objects one by one for processing.

This class uses a Windows timer event to process the events in the queue. One event (if the queue is not empty) is processed for each timer event.

All events occurring in the home network are logged into all the event log(s) attached to the `EventManager`. This class maintains a set of `EventLog` object pointers. Pointers to objects that are derived from `EventLog` can then be attached to this set. Examples of derived event logs are `EventLogFile` (logs into a text file) and `EventLogWindow` (logs into a window).

Each event is processed differently, depending on the type of event. If the event is appliance-related, the status of that appliance is updated. For example, when an appliance is switched on, the status of that appliance is set to 'on'. The appliance, in turn, updates all its views (see `ApplianceView` classes).

Any appliance may have one or more timers (see `ApplianceTimer` class) associated with it. The `EventManager` class is responsible for checking each timer and switching on the associated appliance when the timer starts and switching it off when the timer ends.

All tasks which should be done based on the occurrence of event are performed by this class. This is because all events pass through this class, so these tasks can be *triggered* upon the arrival of the desired event.

Security tasks are an example of this. Security-related appliances (motion detectors, door and window detectors, etc) are monitored when the system is armed. When the events for these devices are received the THC flashes all lights and sounds a siren.

Housekeeping tasks are also accomplished this way. For example, when the telephone light flashes, all radios are switched off. Similarly, Energy management tasks monitor the thermostat appliance and switch the heater on or off, whichever is appropriate.

Other tasks can be *hooked* into the `EventManager` class in similar ways. Any appliance can be monitored and tasks can be triggered based on events on the appliance.

Table 12 describes a specification of Class EventManager. Table 13 shows an abstract implementation of the class using an object-oriented programming language, C++. Full listings of the source codes are not available in this report.

Class Name:	**EventManager**
Function:	Processes and dispatches all events
Attributes:	Events
	`eventlogs`
	`owner`
	`intruderTimer`
	`intruderTimerCount`
	`securityTimer1`
	`securityTimer2`
	`securityTimer1Count`
	`securityTimer2Count`
Operations:	`Start`
	`Stop`
	`AddEventLog`
	`RemoveEventLog`
	`OnAllUnitsOff`
	`OnAllLightsOn`
	`OnAllLightsOff`
	`OnApplianceOn`
	`OnApplianceOff`
	`OnLightBright`
	`OnLightDim`
	`ProcessEvent`
	`LogEvent`
Used by:	`PowerLineController`
Contains:	Unit, TQueueAsDoubleList, TSetAsVector
Inherited by:	None
Derived from:	None

Table 12: Class Specification of EventManager

```
// EVENTMGR.H
// EventManager class
class EventManager {
public:
   EventManager(Unit *owner);
   ~EventManager();
   void Start();
   void Stop();
   void AddEventLog(EventLog *eventlog);
   void RemoveEventLog(EventLog *eventlog);
   void OnAllUnitsOff(AllUnitsOffEvent *e);
   void OnAllLightsOn(AllLightsOnEvent *e);
   void OnAllLightsOff(AllLightsOffEvent *e);
   void OnApplianceOn(ApplianceOnEvent *e);
   void OnApplianceOff(ApplianceOffEvent *e);
   void OnLightBright(LightBrightEvent *e);
   void OnLightDim(LightDimEvent *e);
   void ProcessEvent();
   void LogEvent(const char *eventdesc, TTime time = TTime());
};
```

Table 13: C++ Representation of Class EventManager

3.5.3 Classes Event

The `Event` classes are a hierarchy of classes which encapsulate the different types of events that occur in the home network. The `PowerLineController` class receives data bytes from the communications port and converts them into an object of the appropriate class derived from the `Event` class. This event object is then placed into the event queue of the `EventManager` class.

Events are encapsulated into `Event` classes in order to hide the details of the data bytes received from the communications port. The only class that will need to deal with these details is the `PowerLineController` class. All the other classes only have to deal with the `Event` objects and therefore the complexity of interpreting the data bytes is hidden from them. This also means that if the format or interpretation of the data bytes is changed in the future, only the `PowerLineController` class needs to be changed.

Here is the `Event` classes hierarchy:

```
Event
    HouseCommandEvent
        AllUnitsOffEvent
        AllLightsOnEvent
        AllLightsOffEvent
    ApplianceCommandEvent
        ApplianceOnEvent
        ApplianceOffEvent
        LightBrightEvent
        LightDimEvent
```

All `Event` objects have a timestamp of when they occurred. This is used by the `EventManager` to display the date and time of occurrence in the event logs. `Event` objects also know how to *dispatch* themselves. Each `Event` object knows which method of the `EventManager` class should process it, so it passes itself to that method for processing.

`HouseCommandEvent` (and all its derived) objects also have information on the house code where that event occurred.

`ApplianceCommandEvent` (and all its derived) objects also have information on the appliance address where that event occurred.

Table 14 describes a specification of Class Event. Table 15 shows an abstract implementation of the class using an object-oriented programming language, C++. Full listings of the source codes are not available in this report.

Class Name:	`Event`
Function:	Encapsulate all events from the home network
Attributes:	`timestamp`

Operations:	`GetTimeStamp`
	`Dispatch`
	`GetDescription`

Used by:	`PowerLineController, EventManager`
Contains:	`TTime`

Inherited by:	`HouseCommandEvent`
	`AllUnitsOffEvent`
	`AllLightsOnEvent`
	`AllLightsOffEvent`
	`ApplianceCommandEvent`
	`ApplianceOnEvent`
	`ApplianceOffEvent`
	`LightBrightEvent`
	`LightDimEvent`

Derived from:	None

Table 14: Class Specification of Event

```
// EVENT.H
// Event class
//

class Event {
public:
   Event(TTime timestamp);
   virtual ~Event();

   TTime GetTimeStamp() const;

   virtual void Dispatch(EventManager *emgr) = 0;

   virtual char *GetDescription(char *buf, int buflen, Unit *unit)
const;

...
};
```

Table 15: C++ Representation of Class Event

3.5.4 Classes EventLog

The `EventLog` classes are a hierarchy of classes that are used by the `EventManager` to log all events occurring in the home network. The `EventManager` maintains a set of `EventLog`-derived objects and logs every event to each of them. Currently, two classes are derived from `EventLog`:

- `EventLogWindow` - logs events in a window on the screen display.
- `EventLogFile` - logs events to the file "`C:\THC\EVENTS.LOG`".

The file "`C:\THC\EVENTS.LOG`" is not purged each time the THC application is run. New logs are appended to it so it just keeps growing. This is done so that logs of previous runs are not lost. The user must purge this file occasionally to prevent it from getting too big.

If there is a need for new event logs, a new class can be derived from `EventLog` and an object of this new class can be created and attached to the `EventManager`'s set of `EventLogs`.

Table 16 describes a specification of Class EventLog. Table 17 shows an abstract implementation of the class using an object-oriented programming language, C++. Full listings of the source codes are not available in this report.

Class Name:	**`EventLog`**
Function:	Record the record log to an output device
Attributes:	`eventmgr`
Operations:	`SetEventManager`
	`Log`
Used by:	`EventManager`
Contains:	EventManager
Inherited by:	`EventLogWindow`
	`EventLogFile`
Derived from:	None

Table16: Class Specification of EventLog

```
// EVENTLOG.H
// EventLog class
//
class EventLog {
public:
   EventLog();
   virtual ~EventLog();

   void SetEventManager(EventManager *eventmgr);

   virtual void Log(const char *eventdesc) = 0;

...
};
```

Table 17: C++ Representation of Class EventLog

3.5.5 Class ApplianceAddress

This class represents an appliance address which consists of a house code and a unit code. This address uniquely identifies each appliance within the home network.

The use of this class makes it easier to deal with appliance addresses. Instead of using house code and unit code separately or as a string, they are encapsulated as one object.

Table 18 describes a specification of Class ApplianceAddress. Table 19 shows an abstract implementation of the class using an object-oriented programming language, C++. Full listings of the source codes are not available in this report.

Class Name:	**ApplianceAddress**
Function:	Encapsulates appliance address - house code/unit code pair
Attributes:	housecode
	unitcode
Operations:	GetHouseCode
	SetHouseCode
	GetUnitCode
	SetUnitCode
	GetString
Used by:	Appliance, EventManager, Unit
Contains:	None
Inherited by:	None
Derived from:	None

Table 18: Class Specification of ApplianceAddress

```
// APPLADDR.H
// Appliance address class
//
class ApplianceAddress {
public:
   ApplianceAddress(char house, unsigned char unit);
   ApplianceAddress(const char *addrstr);

   char GetHouseCode() const;
   void SetHouseCode(char house);

   unsigned char GetUnitCode() const;
   void SetUnitCode(unsigned char unit);

   char *GetString(char *addrstr) const;

   int operator == (ApplianceAddress arg) const;
   int operator != (ApplianceAddress arg) const;
...
};
```

Table 19: C++ Representation of Class ApplianceAddress

3.5.6 Classes Appliance

There are two classes that represent appliances:

- `Appliance` - This includes the attributes that describe an appliance, including its address and name, and the operations to switch an appliance on and off, and to check whether it is on or off.
- `LampAppliance` - This class inherits from `Appliance`, so it has everything an `Appliance` has with the addition of operations to dim and brighten the lamp.

These classes send a command to the `PowerLineController` class for each of the operations on it (on, off, dim, bright). The PowerLineController queues these commands until the home network is ready to receive them.

The `EventManager` class updates the on/off status of an `Appliance` or `LampAppliance` object whenever it receives an appliance on/off event from the `PowerLineController` class.

Each `Appliance` maintains a set of `ApplianceView`-derived objects. Whenever its on/off status is updated it also informs all its views to update themselves. All appliance views attach themselves to the appliance they display so they can be updated.

Appliances in the home network are defined in the home configuration file (see section `3.4`). Each appliance definition has the following format:

```
<appliance type>, "<appliance name>", "<appliance address>"
```

where:

`<appliance type>` - This is either `A` or `L` for appliance or lamp appliance, respectively.
`<appliance name>` - This is the descriptive name of an appliance.
`<appliance address>` - This consists of the house code and the unit code of an appliance.

Table 20 describes a specification of Class Appliance. Table 21 shows an abstract implementation of the class using an object-oriented programming language, C++. Full listings of the source codes are not available in this report.

Class Name:	`Appliance`
Function:	Represents an appliance

Attributes:	`name`
	`address`
	`isOn`
	`owner`
	`views`

Operations:	`GetName`
	`GetAddress`
	`SetOwner`
	`SwitchOn`
	`SwitchOff`
	`SetOnStatus`
	`IsOn`
	`GetType`
	`Dim`
	`Bright`
	`AddView`
	`RemoveView`

Used by:	`EventManager, PowerLineController, Unit, ApplianceView`
Contains:	`ApplianceAddress, Unit, TSetAsVector`
Inherited by:	`LampAppliance`
Derived from:	None

Table 20: Class Specification of Appliance

```
// APPLIANS.H
// Appliance Module classes.
//
class Appliance {
public:
   Appliance(const char *name, ApplianceAddress address);
   virtual ~Appliance();
const char *GetName() const;
   ApplianceAddress GetAddress() const;
   void SetOwner(Unit *owner);
   void SwitchOn();
   void SwitchOff();
   void SetOnStatus(int ison);
   int IsOn() const;
   virtual char GetType();
   virtual void Dim();
   virtual void Bright();
   void AddView(ApplianceView *view);
   void RemoveView(ApplianceView *view);
   // This is needed so that we can compare two Appliance objects
   int operator == (const Appliance &arg) const;
...
};
```

Table 21: C++ Representation of Class Appliance

3.5.7 Class ApplianceTimer

This class is used to represent a time range of when an `Appliance` should be controlled. This class consists of a time range and a pointer to the `Appliance` object where this timer should be applied. More than one `ApplianceTimer` object may be applied to any `Appliance`.

Objects of this class are processed by the `EventManager` class during its processing of events (as described in Section 3.5.2).

`ApplianceTimers` are also defined in the home configuration file (Section 3.4). Each appliance timer definition has the following format:

```
T, "<appliance address>", <on hour>, <on min>, <off hour>, <off min>[, <days>]
```

where:

T - This means that the line is an appliance timer definition.
<appliance address> - This consists of the house code and a unit code of an appliance.
<on hour> - This is the hour (0 - 23) when the appliance will be switched on.
<on min> - This is the minute (0 - 59) when the appliance will be switched on.
<off hour> - This is the hour (0 - 23) when the appliance will be switched off.
<off min> - This is the minute (0 - 59) when the appliance will be switched off.
<days> - This is optional and indicates which day(s) of the week the timer is effective. Any day(s) of the week can be specified. If this is not present the timer is effective every day.

Table 22 describes a specification of Class ApplianceTimer. Table 23 shows an abstract implementation of the class using an object-oriented programming language, C++. Full listings of the source codes are not available in this report.

Class Name:	`ApplianceTimer`
Function:	Automated timer for an appliance
Attributes:	`appliance`
	`onTimeH`
	`onTimeM`
	`offTimeH`
	`offTimeM`
	`dayofweekmap`
Operations:	`IncludeDayOfWeek`
	`ExcludeDayOfWeek`
	`ExcludeAllDays`
	`CheckTime`
Used by:	`EventManager`
Contains:	`Appliance`
Inherited by:	None
Derived from:	None

Table 22: Class Specification of ApplianceTimer

```
// APPLTIME.H
// Appliance Timer class
//

class ApplianceTimer {
public:
   ApplianceTimer(Appliance *appliance, int onTimeH, int onTimeM,
      int offTimeH, int offTimeM);

   void IncludeDayOfWeek(DayTy dayofweek);
   void ExcludeDayOfWeek(DayTy dayofweek);
   void ExcludeAllDays();

   void CheckTime(int hour, int minute, DayTy dayofweek);

   int operator == (const ApplianceTimer &arg) const;

...
};
```

Table 23: C++ Representation of Class ApplianceTimer

3.5.8 Classes ApplianceView

These classes are used to provide a visual display of an `Appliance`'s status. They are also used to allow the user to operate or manipulate an `Appliance`. Each `ApplianceView` visually represents exactly one `Appliance` object. However, each `Appliance` may have any number of `ApplianceViews`.

Whenever the user issues a command to an `ApplianceView`, the `ApplianceView` calls the appropriate member function in the `Appliance` object it represents.

Currently, there is only one class derived from `ApplianceView`. This is the `ApplianceWindow` class. This class displays a graphic icon on the screen which visually illustrates the status of the appliance. When the user clicks on the icon with a mouse, a popup menu is displayed, giving the user some options to operate on that appliance.

New classes may be derived from `ApplianceView` to visually represent an appliance in different ways.

Table 24 describes a specification of Class ApplianceView. Table 25 shows an abstract implementation of the class using an object-oriented programming language, C++. Full listings of the source codes are not available in this report.

Class Name:	`ApplianceView`
Function:	Visual representation of an appliance
Attributes:	`appliance`
Operations:	`GetAppliance`
	`SetAppliance`
	`Update`
Used by:	`Appliance`
Contains:	`Appliance`
Inherited by:	`ApplianceWindow`
Derived from:	None

Table 24: Class Specification of ApplianceView

```
// APPLVIEW.H
// Appliance View base class
//

class ApplianceView {
public:
   ApplianceView(Appliance *appliance);
   virtual ~ApplianceView();

   Appliance *GetAppliance() const;
   virtual void SetAppliance(Appliance *appliance);

   virtual void Update() = 0;

...

};
```

Table 25: C++ Representation of Class ApplianceView

3.6 Discussion

3.6.1 Microsoft Windows

Using a graphical user interface (GUI) like Microsoft Windows makes any system easier to learn and use. In a home automation system, a GUI is even more beneficial because the home and its appliances are best represented on the screen as graphical objects that can be manipulated by the user. Displaying appliances this way makes it obvious what they are and the state they are in (on, off, etc.). A pointing device such as a mouse allows easy navigation through the system. By simply pointing on an appliance icon and clicking on it, a menu of allowable functions for that appliance is displayed.

The ability to do background processing (or pre-emptive multi-tasking) in applications is not directly supported in Windows 3.11. This has been achieved using timer events. They, however, are not entirely reliable because timer events are low priority messages which will not be processed if the system is busy doing some other tasks. Because of this it is possible that the THC application may not be able to monitor some events if another application running on the same PC goes into a tight loop for a long period of time. In future versions of Windows, this problem will be solved because of its direct support for pre-emptive mul-titasking.

3.6.2 Object-Orientation and C++

The use of object-oriented techniques in the design and development of the THC greatly eased development and further evolution of the system.

The system was developed in C++ (Borland C++ User's Guide, 1993; Borland C++ Programmer's Guide, 1993), arguably the most widely used object-oriented language. Because of its popularity, there are many development tools available for it, most important of these being class libraries, also known as *application frameworks*. As the name suggests, class libraries are a library of reusable and extendable classes ready to be used for the development of applications.

Each class library has an area of specialisation or domain. In the development of the THC, a class library specialising in the Windows user interface elements was used. Object Windows Library (OWL) is a rich set of classes for creating Windows interface elements (Borland ObjectWindows for C++, 1993). The use of a class library means that the developers can focus more on the application part of the work instead of on the programming of the user interface. Previously, in this sort of development, the developers have to spend much time in the details of the low-level programming of the user interface, which is quite complex. Now, with a class library, time is not spent in the details of Windows programming and developments have concentrated on the home automation aspects of the application.

Initially, the programme was written using an BRANZ-developed application framework known as RRView. This was because at the time of the initial development, the application frameworks which came with Borland C++ programming language were still in their early versions and did not provide a good environment for the GUI development. However, with Borland C++ 4.0, its application framework, OWL, is much improved and provides a good framework for building Windows programs.

The conversion from RRView to OWL did not create any problem due to the object-oriented nature of both RRView and OWL. The use of OWL confirmed that it was much improved. OWL 2.0 contains more classes than OWL 1.0, i.e. it encapsulates many more applications programming interfaces (API). Thus, developers have access to more standard classes.

Another area which benefited from object-oriented techniques is program evolution. A system designed and developed using object-oriented techniques is more easily maintained and extended. The concept of *encapsulation* protects a class from being affected by changes in other parts of the application. The concept of *inheritance* allows the developer to add new classes in a way that reuses the functionality of the inherited class. Lastly, *polymorphism* allows different derived classes to perform their own special behaviour without the user of the class being concerned with the details.

3.6.3 Home Communications Network

At present the home communication network is based on the X-10 technology (Dechapunya, 1992). One of the problems is that it does not yet provide full two-way communication. Standard modules such as a lamp module only receive signals and do not transmit them. This is satisfactory as long as users use remote or any X-10 controllers or PCs to control/automate their homes. The manual operation (local control) of these modules will not transmit X-10 signals. Thus, EventManager will not be able to process the event.

The present design has evolved using one PC interface as both transmitter and receiver. The system may improve if two PC interfaces are used, one to transmit and one to receive.

The hardware device currently used to interface with the home network has a limitation. When it is swamped with events from transmitter(s) in the home network (e.g. a flashing light), it does not become available for transmitting commands until all the events have been received. In this situation, this class is unable to transmit any pending commands, rendering the THC application in a *"disabled"* state. One solution to this problem is to have separate devices for transmitting and receiving. A better solution would be to use a device that can support concurrent two-way communication with the home network (*full duplex*).

Another limitation is the use of a Windows timer to process events in the background. The Windows timer event is a low priority message that may not be sent when the system is busy processing in some other application, or even in other parts of the THC. This will result in the communications port not being able to be checked for an extended period of time, resulting in loss of data. The solution to this is to have a more reliable alternative to background processing. An example of this is a Windows *Virtual Device Driver (VDD)*, which is interrupt driven and runs at a higher priority than Windows applications. However, VDD's are difficult to write because they have to be written in assembly language and require intimate knowledge of the hardware and Windows system level programming.

3.6.4 User Interface

The second design of user interface for appliance control is very successful in terms of being user friendly. The representation of appliances as active objects makes the appliances available to the users at any time. The results have showed the differences between dialog-based appliance control and the object-based appliance control.

The design of the dialog-based appliance control is based on the dialog structure of the Windows architecture. This is a normal method of utilising a Windows environment. The design of object-based appliance control is based on the concept of treating each appliance as an object. Each object represents an actual appliance. The representation has been realised with the powerful object-oriented programming and real-time interaction between the computer and the home communications network. In summary, object-oriented methodology has been extended to represent real automated objects.

3.6.5 Appliance Control

The THC is designed so that all appliances are always accessible to the user. Visually, all appliances are displayed on the screen in such a way that their status is obvious. The user can readily control and operate any appliance at the click of the mouse. This makes the system very easy to learn and use.

The THC appliance control is designed to monitor the home network at startup and to continue to do so while the system is running. This allows the system to keep track of the state of all appliances and display them appropriately. This is very important because the appliances can also be operated outside the PC, using other devices (e.g. hand-held remotes). Aside from this, some security-related appliances (e.g. motion detectors) can transmit signals to the home network at any time, and the system needs to react to these signals immediately by performing security-related tasks.

The system needs to be informed of these operations and signals so that it can update the display of all appliances and to perform tasks which are based on the occurrence of these operations and signals.

The home network is monitored in the background so that the user can still use the system without interfering with the monitoring process.

4.0 CONCLUSIONS

The home automation models, and software classes, developed in 1993-1994 have been extended to housekeeping applications. These applications are mainly the control and automation of lights and appliances.

Object-oriented technology (OO methodology, OO programming language, and applications framework) contains a rich environment that makes developing Windows applications less complex. The complexity of GUI codes is encapsulated in an application framework. The result is an increase in productivity.

The re-engineering of the user interface has been completed. The user interface is now changed from a DOS, menu-based environment to a Windows, Graphical User Interface environment. The new user interface provides a much better environment than the previous DOS menu environment. Users are now in control of the applications. The new design means that all appliances are always accessible to the user. The user can readily control and operate any appliance at the click of a mouse. This makes the system very easy to learn and use.

The design of the appliance control and automation is based on the concept of treating each appliance as an object. Each object represents an actual appliance. The representation has been realised with powerful object-oriented programming and real-time interaction between the computer and the home communications network. The result is an innovative user interface allowing the users of the system to interact with appliances on the screen.

REFERENCES

Borland C++ User's Guide, 1993. Borland International, Inc., Scotts Valley, CA.

Borland C++ Programmer's Guide, 1993. Borland International, Inc., Scotts Valley, CA.

Borland ObjectWindows for C++, 1993. Borland International, Inc., Scotts Valley, CA.

Box, D., 1994. Starting Windows. C++ Report,Vol 6, No 5, June 1994, pp. 53-58.

Comaford, C., 1994. A Guide to Controls and Window Types. PC Magazine, May 31, 1994, pp. 299-304.

Dechapunya, A.H., 1992. Standards For Communications Networks In The Home. Building Research Association of New Zealand, Miscellaneous Report, Judgeford, New Zealand.

Dechapunya, A.H., 1993. Object-Oriented Methodology for Home Automation Applications. Building Research Association of New Zealand, BRANZ Study Report SR52, Judgeford, New Zealand.

Dechapunya, A.H., 1994. Object-Oriented Programming for Home Automation Applications. Building Research Association of New Zealand, BRANZ Study Report SR56, Judgeford, New Zealand.

Leinfuss, E., 1994. GUIs Reap Productivity Rewards. Computerworld New Zealand, February 14, 1994, pp. 19-23.

Linthicum, D.S., 1994. Life After DOS. PC Magazine, May 31, 1994, pp. 203-237.

Mitchell, E., Becker, P., Dlugosz, J., Finnell-Fruth, C., Free, G., Fruth, R., Herring, B.D., and Schulmeisters, K., 1992. Secrets of the Borland C++ Masters.SAMMS Publishing Carmel, Indiana 46032 USA, pp. 730.

Perry J. P., 1993. Your Borland C++ Consultant SAMS Publishing Carmel, Indiana 46032 USA. pp. 449.

Rimmer, S., 1994. Multimedia Programming for Windows. Windcrest/McGraw-Hill, New York, pp. 370.

Roetzheim, W., 1992. Programming Windows With Borland C++. Ziff-Davis Press, Emeryville, California, pp. 464.

Tetewsky, A.K., 1994. GUI Development for Real-Time Applications. Dr. Dobb's Journal, June 1994, pp. 28-41.

Walrath, K. and Hayden, R., 1994. The Philosophy of Designing a GUI. Object Magazine, July-August 1994, pp. 28-44.

Object-oriented graphical user interface for housekeeping applications.
DECHAPUNYA, A.H ; CRISOSTOMO, R.P.
Aug 1995 33783

BRANZ

THE RESOURCE CENTRE FOR BUILDING EXCELLENCE

BRANZ MISSION

To be the leading resource for the development of the building and construction industry.

HEAD OFFICE AND RESEARCH CENTRE

Moonshine Road, Judgeford
Postal Address - Private Bag 50908, Porirua
Telephone - (04) 235-7600, FAX - (04) 235-6070

REGIONAL ADVISORY OFFICES

AUCKLAND
Telephone - (09) 524-7018
FAX - (09) 524-7069
118 Carlton Gore Road, Newmarket
PO Box 99-186, Newmarket

WELLINGTON
Telephone - (04) 235-7600
FAX - (04) 235-6070
Moonshine Road, Judgeford

CHRISTCHURCH
Telephone - (03) 366-3435
FAX - (03) 366-8552
GRE Building
79-83 Hereford Street
PO Box 496