Transactions on Information and Communications Technologies vol 20, © 1998 WIT Press, www.witpress.com, ISSN 1743-3517

Techniques for Distributed Intelligence and Internet Connectivity on Home Automation Networks.

Peter M. Corcoran.
Dept. of Electronic Engineering, University College, Galway;
pcor@eek.ucg.ie

Dana Vasiloaica.
Dept. of Electronic Engineering, University College, Galway;
dana@eek.ucg.ie

Abstract

Home Automation introduces a range of interesting problems in distributed intelligence. In a home network consumer appliances are interconnected in an arbitrary manner by the end user. Each appliance must be capable of establishing its own network connection, configuring itself without user intervention and of interacting with other unknown devices on the home network.

Furthermore, if a home network is to be connected to an external wide-area-network, such as the Internet, then the collective group of intelligences which form the home network must be marshalled into a coherent entity. This can be most easily achieved through a three-tier software architecture for home gateways [1]. A brief overview is of this architecture is presented.

1. Introduction

The Consumer Electronic Bus (CEBus) is a multi-media LAN standard developed for home automation applications [2]. In earlier work we discussed some of the issues involved in providing access to a CEBus network from a conventional Web browser and Java virtual machine JVM [4]. One of the key features of CEBus is its object model for device inter-operability, the Common Application Language (CAL).

In this paper we describe some of the techniques which are employed to enable consumer appliances with relatively low processing capabilities to act as independent entities on a home automation network. Much of this capability is achieved through an application layer implementation of the Common Application Language as originally described in IS-60 [5]. CAL provides mechanisms for a consumer appliances to maintain an internal description of its strucure and organization which may then be queried by other home network appliances.

The Common Applications Language, or CAL is an object-oriented description language developed to support home automation applications. This object-oriented methodology offers the best means of understanding the complex interaction between devices, controls, sensors and controllers present in an embedded control environment. CAL was designed as a robust, general purpose command language not dedicated to any particular function within the Application Layer. It provides a language for controlling embedded electronic hardware devices and allocating and sharing resources. Further, CAL provides an object-based structure for modeling electronic systems in terms of devices, control-contexts within a device and objects and instance-variables within each context.

More recently a new industry initiative in the US has focused on adopting CAL as a more generic standard across a range of emerging new technologies in home networking. This *Home Plug 'n Play* (HPnP) initiative seeks to adopt CAL as a common interface to emerging and established technologies such as IrDA, IEEE 1394, Powerline CEBus and a range of wireless network technologies. In this paper we look in detail at HPnP and summarise how it extends and enhances the distributed intelligence capabilities of the original CAL specification. We shall also consider how it provides much of the intelligence necessary to support the requirement of networked control systems at the device and sub-system level.

We also describe how local control/automation network may be bridged to the Internet, or an equivalent WAN. As we shall discuss shortly, we feel it is neither practical nor desirable to attempt a direct mapping of network addresses and protocols between these two distinct networks. The functionality and operation of a local control network are somewhat different from those of a wider bandwidth WAN network. Thus we shall focus on the concept of an

intelligent gateway whose function is to manage, broker and integrate network traffic between these two distinct categories of networks. This, in turn, integrates a higher level of system intelligence which allows the functionality of a local HPnP network to be accessed and managed at higher level of abstraction.

In this paper we also describe an object-oriented software framework to facilitate Internet access to home networks which utilise CAL and HPnP. This software consists of (i) a TCP/IP bridge from the home network, (ii) a server application and dynamic data structures which provide a real-time representation of a the state of the individual devices/ intelligences which comprise the distributed home network and (iii) a client application which provides end-user access to the home network. Communications activity on the home network is continually monitored by the TCP/IP birdge and interpreted and the appropriate data structures updated accordingly by the server application. The client application layer interacts with the server to monitor changes in the state of devices on the home network, or to initiate such changes.

In addition the creation of "virtual" appliances is supported. These "virtual" appliances can interact with real appliances on the home network. This allows the creation of "new" device personalities which are composed of a combination of service and interface functionality derived from and controlled through interaction with several real-world appliances.

Internet functionalily and a range of interface services are provided via an integral HTTP server. A reference implementation has been developed using conventional desktop PC hardware, although the underlying system software can be easily ported to other hardware platforms to support integration into set-top boxes and related consumer electronic systems. The system has a powerline interface to the home network and Internet connectivity may be achieved through standard ethernet or telephone links.

The functions of such a home gateway can be sumarised as follows:

- acts as ***broker*** for WAN (outside world) access to local network (home).
- provides local (home) access to WAN ***services & resources.***
- implements ***security & encryption*** services.
- provides ***secure remote access*** across the WAN (outside world) to local network (home).

One service which we see as very significant in the context of home automation networks is the ability to access and download user-interface (UI) modules for home network appliances. This issue of providing a suitable user-interface to home automation networks is described elsewhere [4].

Note, that although we have taken the CEBus networking standard as the basis for the implementation of our gateway technology, it is not confined to operating with CEBus networks. In fact any home automation network which offers support for the Common Applications Language (CAL) can be supported by our gateway technology.

2. HPnP Architecture for Inter-operability

The HPnP architecture for inter-operability may be described in five different layers of constructs. The most basic layer begins with the Basic CAL building blocks. These constructs are documented in EIA [5]. They include the concepts of addressable devices, contexts, context numbers, a foundation set of objects that contain instance variables, and a reporting mechanism. It also embraces a required set of transport-level services. In the second layer, we add the HPnP building blocks of a subsystem, status and listener objects, sensors, alarms, and a subsystem for developing and sharing house mode information. At the third level, we find the mechanisms available for creating inter-operability among HPnP devices and subsystems. These include default binding, loose coupling, dynamic context numbers, HPnP broadcast of status information, and state vectors. The fourth layer describes the use of tightly coupled objects for intra-subsystem communications. A final layer adds other elements that are essential for inter-operation. This is illustrated in Table 1 below.

Table 1. *HPnP Constructs*

HPnP Constructs	Elements
CAL Building blocks:	Devices, contexts, context numbers, objects, instance variables, CAL reporting, HPnP broadcast and directed messaging
Building blocks used by HPnP	Subsystems, status & listener objects, sensors, alarms, house mode.
Inter-operability constructs for subsystem to subsystem (Inter-subsystem) communications	Default binding, loose coupling, dynamic context numbers, broadcast of status information, state vectors, HPnP zoning
Inter-operability constructs for tightly coupled (Intra-subsystem) communications	Tight coupling, installation tools
Other requirements essential for inter-operation	Start-up, configuration, resource management, authentication and encryption and transport requirements

2.1 Basic CAL Building Blocks used in HPnP

Using the CAL building blocks of contexts, objects, and instance variables, various levels of products may be built. These may be sensors or actuators, single-function devices, or devices that contain many functions. To inter-operate in a Home Plug and Play manner with other CAL entities, these products need to be designed using the CAL standard and with the contexts, objects, and instance variables as specified in [ref. HPnP standard].

The EIA 600.81 (CAL) standard (section 8) provides a standard set of object definitions. Multiple of these objects may be combined in contexts to be used for specific functions. Multiple contexts may be provided in one product to

perform the desired combinations of functions. To help clarify these concepts the following definitions are provided:

object - A CAL term used to define a single control function within a context. For example, an Audio context contains a Gain Object**.**

context - A group of one or more CAL objects representing a common device function. Several of these contexts may be present in a single device.

device - A mechanism that exposes state and control variables through a home network using the CAL protocol. Devices might be stand-alone hardware devices or might be implemented in software on a PC. A device is a container for a set of CAL contexts that collectively receive messages addressed to the same Transport layer address. This address may be a unique system or unit address, one of many group addresses, or a broadcast address.

2.2 HPnP Building Blocks

The first HPnP building block to consider is the subsystem construct. A subsystem is a device that manages a major set of functions within a home control system. Examples of subsystems include: security system, lighting system, environmental control system, home entertainment system. A subsystem contains a set of CAL contexts that collectively are assigned responsibility for a portion of a control region. A subsystem typically consists of a subsystem controller and other hidden or exposed functions. Exposed functions are visible to other subsystems and devices on the network.

HPnP subsystems may reside in a single device or may be spread over many devices. A device may contain all or parts of multiple subsystems. A subsystem may be packaged in multiple devices. This is illustrated in Figs 1(a), (b) and (c). For example, a user interface on a security subsystem may be packaged as a wall panel, whereas a security system controller may be packaged for mounting in a utility area for connection to a large number of door and window sensors. Both devices may be visible on a home network and still be part of the same subsystem. The information flow between multiple devices such as these may be extensive and detailed. Depending on the design, this information may or may not need to be shared with other subsystem. Also, depending on the design, the binding of these types of devices may or may not require a professional installation tool.

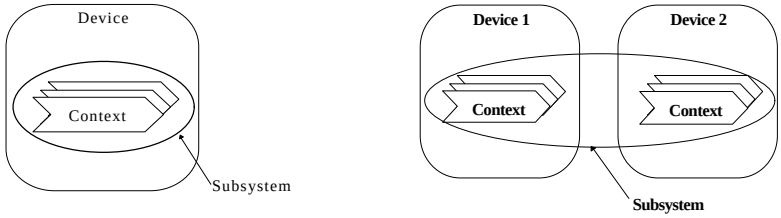

Fig 1(a) *One HPnP Sub-*

system per CAL Device

Fig 1(b) *One Subsystem May Span Several Devices.*

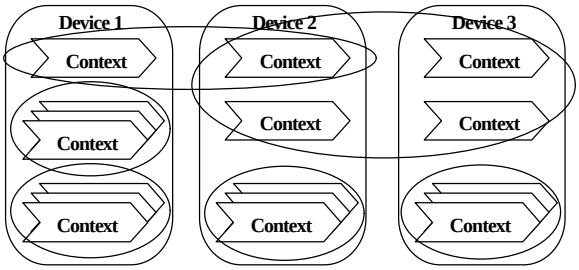

Fig 1(c) *Illustration of Six Subsystems Contained in Three Devices*

One reason for implementing subsystems is to enable lower cost products. Few low-cost devices (such as sensors) can bear the cost of a full network connection. The subsystem concept allows multiple functions to be aggregated in a single network connection. Subsystems may include object models of many types of home control functions. An example could be an outdoor temperature sensor that is directly wired to an environmental controller. The environmental controller may share this outdoor temperature with other HPnP subsystems by providing a context that contains an object model of the sensor.

Subsystems may also contain status and state information for their control region that may be shared with other subsystems. For example, a security subsystem may share its arm/disarm state with a lighting subsystem so that the lighting subsystem could provide exit lighting and a lived-in-look function. When a HPnP product implements status or state information provided by this specification it will be able to inter-operate with other HPnP products listening to the same information.

Standard Modeling Construct—Home Plug and Play specifies sets of CAL constructs in which subsystems can publish their status, listen to other subsystem's status, and request status changes in other subsystems. This is accomplished both by using objects developed by industry groups and by using a new vector notation. This construct is intended for replication throughout HPnP device construction and is necessary for exchanging inter-operable HPnP vectors. Definitions of these object types follow:

- **Status objects** are any objects that uses the CAL reporting mechanism to convey current state or parameter information and whose report header and destination address is configured to bind with a listener object.
- **Listener objects** are objects that receive reports from status objects of one context and have logic to modify the status in objects of other associated contexts based on the value(s) received. A listener object does not report.

- **Request objects** are any objects that send requests to change the value of a status object. A request object may change the value of a status object by sending a request.

Shared sensors are devices that contain a sensor context designed to share the sensor information with subsystems and devices.

When a sensor context reports to the HPnP broadcast destination, the information is delivered to any device that wishes to listen. To receive the sensor information the listening device must implement a context containing a sensor listening object. If multiple sensors of the same type are available in the home, a listening context may decide to listen to a particular sensor by adjusting its context number to match that of the desired sensor.

HPnP Alarms—Another HPnP building block are alarms. Any control region of a home may develop alarms. This building block provides a common way to share these alarms with other subsystems in the home. Note that HPnP alarms are not intended to be compliant with approval bodies' requirements and therefore must not be relied upon to support alarms in critical life-safety applications.

House Mode Context—A feature of the HPnP subsystem communications interaction is a new context class for expressing house mode. This context provides a method for representing the major activities of the home occupants to all HPnP subsystems. By receiving HPnP broadcast information about occupancy and occupant activities, all listening subsystems can adjust their behavior per their own design. This provides a basic level of integration and harmonization in the simplest of home control systems.

2.3 Inter-operability

Subsystems may wish to interoperate within their own subsystem or with other subsystems. The CAL context model supports inter-operation between contexts within a subsystem (intra-subsystem) and across subsystem or subsystems (inter-subsystem). The concept is illustrated in Fig 2.

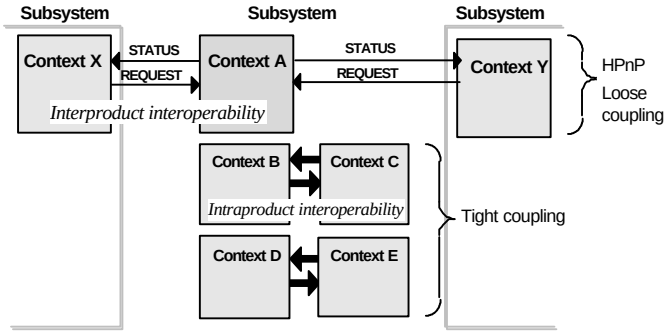

Fig 2: *The basic model of inter and intra subsystem inter-operability.*

Each industry context group in this document (Sections 5 and 6) contains the necessary contexts for inter- and intra-operation of almost all types of equipment and/or functions within a subsystem. This provides the details for subsystem component operation to be handled within the subsystem and for high-level subsystem operation to be controlled and monitored by other subsystem.

2.3.1 Intersystem Inter-operability

The loose coupling and default binding aspects of HPnP, explained below, deal with inter-subsystem inter-operability by providing contexts within each subsystem that are loosely coupled to contexts in other subsystems for exchange of general request, status, and state information.

Each subsystem deals with inter-subsystem communications by providing one or more contexts that specifically provide status and possibly control information to and from one or more contexts in other industry groups. This allows monitoring and control of subsystems on a whole-house level and is the basis for the majority of HPnP operation.

Default Binding—A key mechanism for inter-operation among HPnP products uses CAL's reporting mechanism. Parameters and state change information to be shared with other products is reported when the information changes by broadcasting to predetermined context and object types. This default report mechanism is supplied in all HPnP products, allowing entry-level inter-operability without customization. Professional installers may modify the reporting mechanisms as desired using CAL commands.

Loose Coupling—Status information from status objects, provided by default binding, is sent to all other interested HPnP subsystems in the home. The status object need not aware of what other subsystems are installed and listening. Interested peer subsystems can listen to information and modify their own behavior per their own unique designs. This loose coupling mechanism allows HPnP products to be incrementally installed by the homeowner. It also allows vendors to design their inter-operable HPnP products without having detailed knowledge of other vendor's products.

Loose coupling requires that the HPnP broadcast address be used as the destination address of a CAL reporting object in the transmitting node. The distinction between destinations is made based on the transmitted CAL Context Class IDs, Context Numbers, and Object Identifiers. A listener may subscribe to this information by instantiating the correct context and object.

2.3.2 Intra-subsystem Inter-operation

In addition to the loose coupling concepts discussed above, HPnP systems may employ reports to specific objects when appropriate. In these cases, the sender needs to know not only the context, object, and instance variable IDs but, also the device address of the receiver. This is some times called **tight coupling** and

can be very useful for reducing HPnP broadcast traffic on a network, and for messages that require immediate acknowledgment.

Using tight coupling, a sending device may transmit its information to a specific receiving device. This requires an installation routine in the transmitter to acquire an address (unit or group) of the destination device. Tight coupling requires that a non-broadcast address be used as the destination address of the CAL reporting object in the transmitting node.

This document also includes the necessary context model information within each subsystem to provide for intra-subsystem inter-operability. Intra-subsystem communication generally relies on tight coupling between contexts because messages are specific to devices working together within the subsystem. This type of communication typically deals with status and control from, for example, the thermostat to the heating/cooling equipment, the light switch to the light, the security sensor to the security panel, and so on.

2.4 State Vector Overview

A major mechanism for HPnP inter-operability is the state vector. This concepts allows high-level state information to be shared by each subsystem so that other subsystems may listen as desired. A subsystem interested in listening to state information from its peers has only to instantiate an appropriate listener object to receive this state information. HPnP specifies that high-level state information provided be abstracted so that listening subsystems need not understand all the details of how a particular subsystem is constructed. As an example, it is desirable to know that an environmental control is operating in a heating mode without knowing how heat is delivered, whether energy is currently being expended, or the energy source.

To allow for evolution in the type of state information that can be broadcast and received, HPnP specifies extensible hierarchical state vectors. The vectors are hierarchical in that each succeeding level provides specialization of the previous level. Depending on the designs of these state vectors, listeners of these subsystems can interpret them only to the level understood or desired. For example, a state vector value describing a user's desire for how a home should be operating might be "Home.Asleep," "Away.Errand" or "Away.Vacation." The top level vector component values of "Home" or "Away," could stand alone, but the further specialization of "Asleep," "Errand," and "Vacation" adds a more precise and useful interpretation for subsystems that can provide more specialized behavior.

State vectors are broadcast to the HPnP broadcast destination address so that all interested subsystems can listen. The state vectors communicated for each subsystem type are different and are specified in state vector tables found in later sections of this specification. The exact meaning of the vectors is subsystem specific and determined by industry working groups.

2.5 HPnP Zoning

A HPnP zone is a physical or logical portion of a control region. To accomplish zoning, various methods for logical and physical association are required. In HPnP, simple logical association is accomplished with standard CAL constructs, including Context classes, Context numbers, Object IDs, and Instance variable IDs. The dynamic Context Number technique described above plays a major roll in HPnP zoning.

In addition to the use of dynamic Context Numbers some low-cost devices (lights and light modules) may rely on the use of group addresses to form associations. HPnP also supports this technique.

Another construct provided in some HPnP subsystems is based a “type_ID” instance variable. This construct further differentiates context classes and allows listeners to automatically find an appropriate status provider.

3. Design Goals for a Home Gateway.

A central theme in our implementation of an Internet/Home-bus gateway is that the system architecture should be lightweight and platform independent. This is particularly important in home automation applications where “cost is king”. Thus although the penetration of desktop PC technology into the home market is increasing on a daily basis we still expect that most homes will not rely on a home PC as the home gateway. Instead we expect that a set-top-box style solution with a low-cost embedded CPU will satisfy this need for most end-users.

Furthermore we see two distinctly different approaches emerging in the field of home automation: one approach is to incorporate TCP/IP networking functionality into even the simplest of consumer appliances. There are several drawbacks to this approach. The current version of the Internet Protocol (IP) allocates 32 bits for addressing. As CEBus also requires 32 bits there is not enough room to map every CEBus address to a unique IP address. Also many embedded processors used in consumer devices are not powerful enough to implement the IP protocol.

The second approach, which we feel is more realistic, is to integrate existing established technologies, such as CEBus/CAL with the TCP/IP infrastructure. With this approach only one device - the gateway server - need be smart enough to implement the IP protocol, and only one IP number need be assigned per home. Current industry initiatives, such as the proposed *Home Plug and Play (HPnP)* standard [6], would appear to support our thinking in this matter.

3.1 Scalability of the Gateway Architecture

The prototype gateway has been developed on a standard desktop PC platform. However to support scalability it has been almost entirely implemented in the Java® language. This should greatly simplify the porting of the gateway to low-end embedded applications such as Set-Top-Boxes and Web-TV appliances.

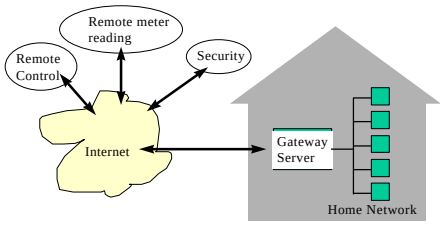

Fig 3: *Home Gateway to the Internet with some typical user services.*

3.2 Independent User Interface Modules

A further strategy to keep the gateway lightweight and scalable is the approach to user-interface issues. These are covered in more detail elsewhere [4], but essentially the gateway interface only supports a simple device browsing service. Independent user-interface elements for each network device are loaded from external network URLs - typically in the form of home -interactive programlets, or HiPlets, as described in [4]. Thus the gateway can support a practically unlimited variety of user-interface components.

4. Software Components & Subsystems

A central theme in our implementation of an Internet/CEBus gateway is that the system software architecture should be lightweight and platform independent. To achieve these goal most of the key system components have been implemented in Java.

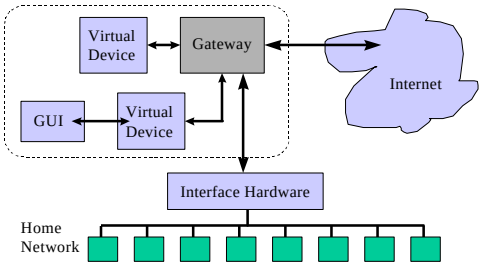

Fig 4: *Overview of Gateway Software and Hardware components.*

The primary components of our system are:

- The Internet/CEBus Gateway Server
- CEBus/CAL Class Library
- Virtual Devices
- Interface hardware

4.1 Internet/CEBus Gateway Server

The Gateway Server is a Java application that acts as an intermediatry between applications and the home automation network. The server communicates to the home automation network via a proprietery interface device, somewhat akin to a conventional telephone modem. However this hardware is somewhat more flexible than a simple modem and can allow *virtual devices* to interact with the home automation network as if they were *real devices* and had direct hardware access to the network. This flexibility has some interesting implications with respect to developing new devices and/or CAL product models.

Applications request services from gateway by establishing a TCP (Transfer Control Protocol) socket and then exchanging a protocol similar to HTTP (Hyper Text Transfer Protocol or World Wide Web protocol) [RFC 1945, RFC 2068]. Using Secure Sockets Layer (SSL) full security can be achieved, which is vital for any remote control or remote metering applications. SSL will ensure that all transactions are encrypted. It also provides a means to authenticate the identity of both parties by using public key cryptography certificates.

A lighter communication protocol using UDP (User Datagram Packets) has also been implemented. The UDP packets to not incur the overhead of creating a TCP socket and thus is more suited to heavy load situations (eg routing packets). However UDP does not provide guaranteed delivery.

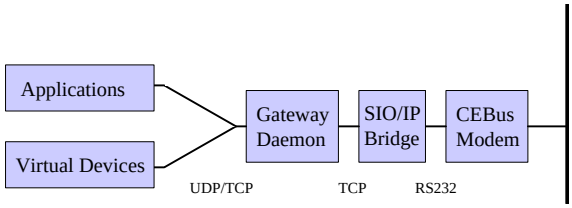

Fig 5: *Structure of the Gateway S/W Components.*

As Java currently does not have a Serial I/O API communications with the CEBus modem is achieved using a simple server program written in C which channels serial I/O to a TCP socket. Java native methods could also be utilized to overcome this limitation.

4.2 Virtual Devices and Networks

In the initial work to implement a prototype gateway one problem we encountered was a shortage of available CEBus compliant devices. Most available devices which employ the CEBus data transport are partly proprietary and as such are not fully compatible with the EIA standard.

To over come this problem and assist in testing and debugging our prototype gateway we set up a second desktop PC with a virtual network of Home Bus devices. Network traffic from this simulated network of devices was broadcast

over the powerline network, appearing to the gateway system as if a real powerline Home Bus network actually existed.

Virtual devices also provide a convenient mechanism to implement a GUI to a home device or sub system. For example a virtual device bound to a GUI interface running on a set-top box could act as the controller to the home security system replacing the normal keypad controller fixed to the wall.

4.3 Virtual Home Bus Network: Class Libraries

A Java class library provides all the necessary tools to construct Home Bus devices and network and implement CAL. The network can be run as a standalone simulation on a computer or bridged to other virtual or real networks. Furthermore the class structure as outlined in this section is amenable to extension to provide a network infrastructure which can, handle a multiplicity of Home Bus networking protocols and physical media and, most importantly, can represent these through a single unified software entity to wide-area networks and a range of client software and user interfaces.

Some of the main classes include: CEBusNetwork, CEBusDevice, CALContext, CALObject, CALIV. These are listed in Fig 6 below with an explanation of the primary function of each.

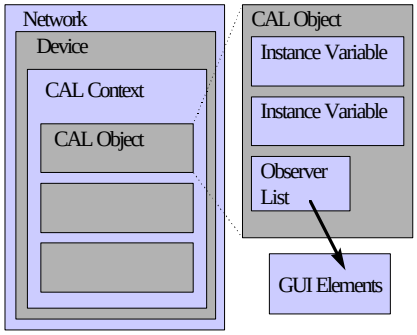

Fig 6: *Network, Device and Context Structures.*

4.4 Code Samples

The following Java code fragment illustrates the creation of a virtual CEBus network of two devices: a lamp and a dimmer switch. The reporting state of the switch is set so that whenever its 'current_value' changes a packet setting the lamp to the appropriate illumination value will be generated.

First a virtual CEBus network is created:

```
// Create CEBus Network
CEBusNetwork mynet = new CEBusNetwork();
```

The next block of code creates a light dimmer switch with house code 1, unit code 1.

```
// Create light  dimmer switch (house 1, unit 1)
//
CEBusDevice myswitch
                        = new CEBusDevice(1,1);
```

It is identified as device model “SW001” from a fictitious “Acme Inc.” vendor.

```
// Create Universal Context  & Node Control
switch_univctx = new UniversalContext();
switch_nodectl = new NodeControl()
switch_nodectl.setIV(“manuf_name”,”Acme
Inc.”);
switch_nodectl.setIV(“manuf_model”,”SW001”);
switch_univctx.addObject(switch_nodectl);
myswitch.addContext(switch_univctx);
```

A lighting context is created with an AnalogSensor object as object #3 according to the CEBus specification.

```
// Create Lighting Context & Sensor object
switch_lightctx = new LightingContext();
sensor = new AnalogSensor();
switch_lightctx.addObject(sensor,3);
myswitch.addContext(switch_lightctx);
```

A lamp is now created with unit code 2.

```
// Create lamp  (house 1, unit 2)
//
Device mylamp = new Lamp(1,2);

// Create Universal Context & Node Control
lamp_univctx = new UniversalContext();
lamp_nodectl = new NodeControl()
lamp_nodectl.setIV(“manuf_name”,”Acme Inc.”);
lamp_nodectl.setIV(“manuf_model”,”LAMP01”);
lamp_univctx.addObject(lamp_nodectl);
mylamp.addContext(lamp_univctx);
//
// Create Lighting Context & Control object
lamp_lightctx = new LightingContext();
lightlevel = new AnalogControl();
lamp_lightctx.addObject(lightlevel,2);
mylamp.addContext(light_lightctx);
```

Next the dimmer switch is configured to send report packets to the lamp. The CAL header is constructed in byte array “calhead”. The report packet is formed by appending the report value to the supplied CAL header. In this case the header instructs the lamp to set its current value (labeled “C”) to the dimmer switch current value.

```
// Set reporting on switch
byte [] calhead = new byte [5];
```

```
calhead[0]=CAL.CONTEXT_LIGHTING;
calhead[1]=2;  // Send to object #2
calhead[2]=CAL.METHOD_SETVALUE;
calhead[3]='C';
calhead[4]=CAL.TOKEN_DELIMITER;
sensor.setIV("report_address",
mylamp.getAddress());
sensor.setIV("report_header",calhead);
```

Finally register these devices with the network and launch the network browser.

```
// Register devices with network
mynet.addDevice(myswitch);
mynet.addDevice(mylamp);
  // Launch network browser
Frame f = new NetworkBrowser(mynet);
f.show();
```

When executed the device browser window (Fig 7) appears on the desk top. A list of all registered devices appears along the top. Clicking on the 'Browse' button for a given device launches it's 'User Interface' (Fig 9).

The device browser provides a graphical representation of all objects in the device grouped by context.

5. System Architecture & Implementation

In this section we briefly describe the gateway architecture and how it supports intelligent UIDevs which interact with the core system daemon, CALNetd and with external real-devices (Rdevs) and internal virtual-devices (Vdevs).

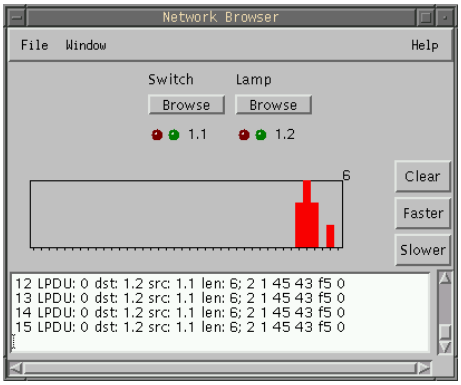

Fig 7: *Network Browser Applet*

5.1 System Hardware

In its present implementation the system is implemented on a desktop PC platform, but as all of the key system software is coded in a platform independent language the core system elements are suitable for down-sizing and porting to the embedded hardware platforms used for Set-Top-Box or Web-TV appliances.

5.2 The Gateway Software Daemons

The software architecutre of the Internet/CEBus gateway is described in detail elsewhere [5].

Briefly a lightweight SIOd daemon continually monitors the traffic on a local powerline network. As devices initiate or receive messages, coded in Common Application Language (CAL) this system daemon provides low-level hardware acknowledgements and responses as required, then passes on the CAL messages to a big-sister daemon, CALNetd.

5.3 The Device Browser Interface

The user interface has two distinct components. Firstly there is a graphical device browser for the local home network. This browser simply shows the list of network devices registered in the local database, in the same way that file management utilities show a list of files in a local filesystem. However when a device is accessed in the browser the user-interface for that devices does not originate from the local system software - instead it is loaded, as an applet, from an HTTP-style universal resource locator (URL).

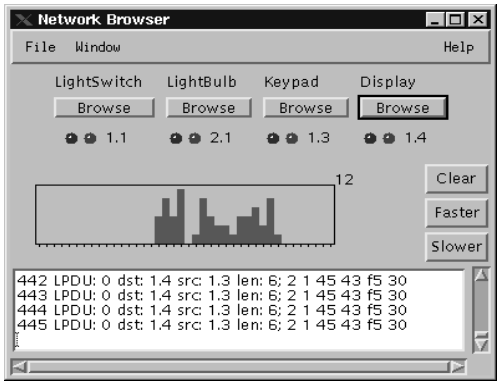

Fig 8: *The Device Browser showing a simple network with 4 devices: a light-switch, light, keypad and display (list memory).*

The browser described above is quite a simple implementation of a device browser. It provides a very useful level of interface for most end-users who only need partial access to the complete functionality of most network devices. However, for the purpose of developing new UIDevs, HiPlets and VDevs it would be very useful to have a more flexible browser which provides detailed access to the complete CAL structure of network devices.

In the interests of completeness we show in Fig 9 such an alternative browser implementation which can provide very detailed access to the CAL structure of a network device. In addition this alternative browser implementation is also aware of many of the standard Device and Context structures defined in [4]. It is somewhat more sophisticated than we would expect to see in a home user's system, providing access to the entire context/object/instance-variable structure of a CAL device. The function of such a browser is to provide an engineering

tool to assist in the development and debugging of new UIDevs and HiPlet interfaces to the home network.

5.4 Software Interfaces to Individual Devices

This scheme allows the local internet/CEBus gateway to provide user interfaces to an unlimited number of different consumer appliances. Furthermore it satisfies the manufactures requirements to be able to customize and differentiate their products by creating unique user interfaces and product features.

To support the development of more sophisticated UI elements we have introduced the concept of grouping UIDevs into interactive container objects known as home-interactive programlets, or HiPlets. This allows several different network devices, or appliances to have their functionality combined and accessed through a single user-interface. At the same time, each device may have own independent interface. Each such HiPlet attaches to a specific CALNetd socket which is mapped onto a local CEBus MAC address. Thus a one-to-one mapping between user interface elements and local network devices is maintained.

5.5 CALNetd Interface Services

As was mentioned above, we are still developing and experimenting with the internal services and structures of CALNetd. The basic services which are currently available include:

- full packet dump
- filtered packet dump
- tx packet/tk_ack packet
- list network devices
- list device state info
- seek device state info
- select TCP/UDP broadcast
- network statistics
- ASDU splitting

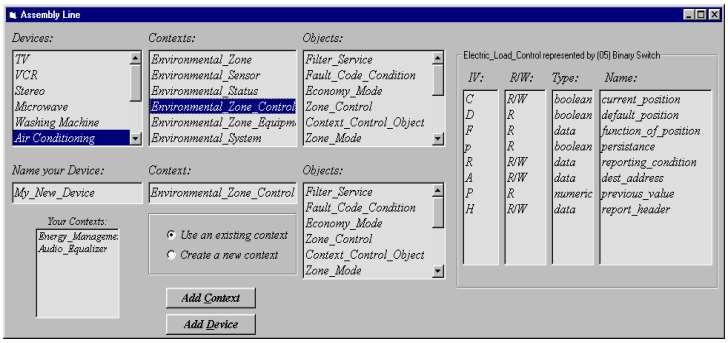

Fig: 9 *An Enhanced Device Browser can provide very detailed access to the CAL Object Structure of an Appliances.*

6. The Gateway Network Architecture

The system elements of our intelligent gateway, as described in the preceeding sections of this paper have led us to consider how best to organize and manage

the different software components which implement the system architecture of a gateway to a home automation network.

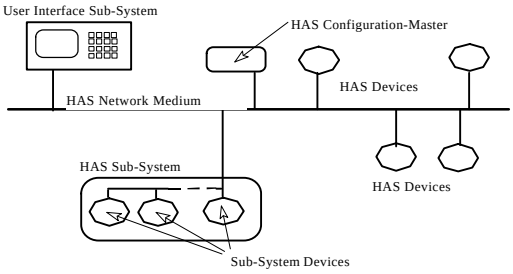

Fig 10: *Conventional Architecture for a Home Automation System Network.*

In Fig 10 we illustrate a conventional architecture where the home network is configured and managed by a dedicated *Configuration Master*. This is typically an embedded system which manages the network according to the specifications of CEBus and CAL. There is little, or no interaction with the end-user and the *Configuration Master* is ofter a specialized and expensive unit.

6.1 Configuration of the Prototype Gateway

In Fig 11 we show a more modern Home Network where a Home PC doubles as both the network *Configuration Master* and acts as a *User-Interface Unit* to the network. Practically all modern automation systems now offer such an option. The PC allows a user to run applications to monitor the network as a whole, and to interact with individual applicances connected to the network.

However we note that a PC is not the ideal means for implementing a home gateway. In particular, a home gateway should be running 24hrs a day, as many domestic applicances and subsystems will be operating at night or when the home owner is not present. Thus a PC is not ideal for performing tasks as the network *Configuration Master*.

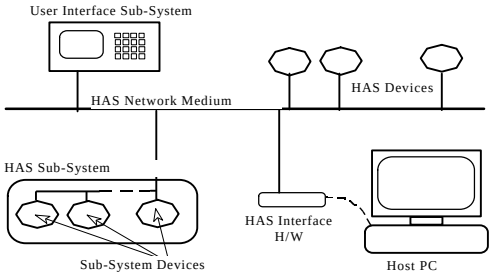

Fig 11: *Home Automation System with PC serving as both Configuration Master and User-Interface Unit.*

6.2 An Embedded Gateway Solution

A much more appropriate gateway solution is illustrated in Fig 12 below. In such a configuration the *Embedded Interface Gateway* may be either a

dedicated unit, or the software infrastructure described in this paper may run on a set-top-box, Java-phone or other *Internet Applicances* with suitable physical interfaces to the HAS.

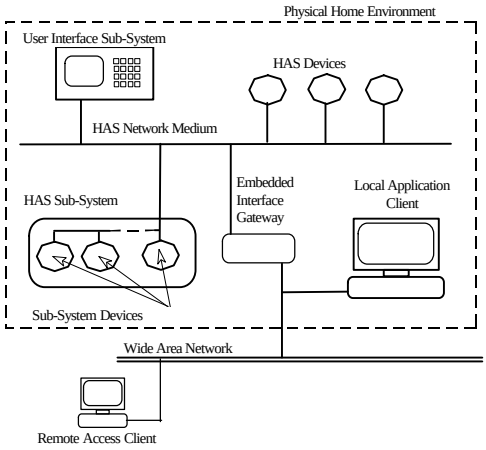

Fig 12: *Home Automation System with an Embedded Interface Gateway; in this configuration the PC is used only as a User Interface subsystem.*

Fig 12, thus represents what we believe will become the standard network architecture for home network gateways in the near future. In this configuration the Home PC remains as a *User Interface Unit* but it is not required to be constantly monitoring and interacting with the home network. Furthermore, we note that in this configuration user access to the home network can be readily achieved from remote locations over the WAN infrastructure within a house, or indeed via an external Internet connection.

Further consideration of this system configuration and the various software components which comprise the gateway leads us to some interesting conclusions as we shall explain in the next section.

6.3 Network Architecture and S/W Components

In Fig 13 the relationships between the system software components of the home gateway, the local home network and the wide area network is illustrated. In particular we note that the software components, SIOd, CALNetd and any system UIDevs/HiPlets all communicate with one another via network sockets. In this configuration, which is employed in our prototype gateway, all of the main components run on the same hardware unit - in our case a standard PC. Applications and UIDevs/HiPlets may run on other *Application Clients*, but all of the key system software components run, as independent *daemon processes* on a single PC.

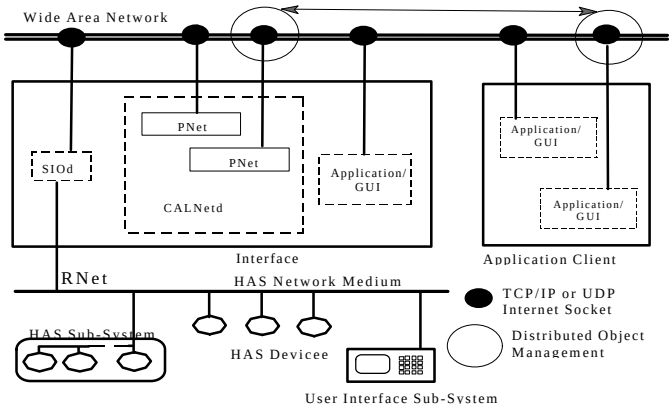

Fig 13: *Diagramatic representation of the relationships between system software components, the Wide Area Network and the Home Network.*

In fact, this is not a requirement for the architecture we have described and there it is interesting to consider the software architecture in more general terms, particularly the potential for separating the system components physically and allowing them to run on separate hardware units, but all connected to the same WAN infrastructure.

6.4 The Three-Tiered Architecture

The considerations descirbed in the last section have led us to formulate the concept of a three-tiered network architechture for the interface between a home network and a wide area network such as the Internet. This is illustrated in Fig 14 where the SIOd runs on a separate hardware unit from the CALNetd, which runs on a *Routing Host*. Application programs, UIDevs and HiPlets run on a third, *User Interface Unit*. This architecture is practical because all of these software components communicate with one another over network sockets.

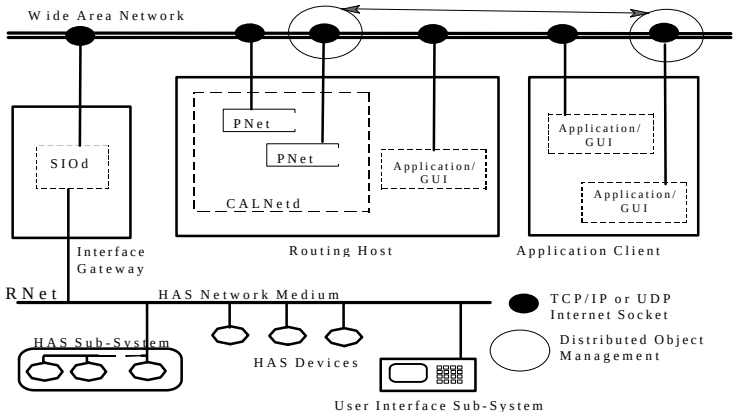

Fig 14: *Three-tiered network architechture for the interface between a home network and a wide area network*

6.5 A Universal AI Interface to Multiple Home Networking Technologies

The three-tiered architecture we have just described has an interesting benefit in the field of home automation where there are several competing standards for networking protocols and also at the physical/transport layer.

In Fig. 15 we show how different home networks can be integrated via multiple SIOd daemons, running on a dedicated "modem-like" hardware subsystem. In this configuration all of the management and routing intelligence is implemented in a common CALNetd which provides a single point of access for external WAN services to the home network environment.

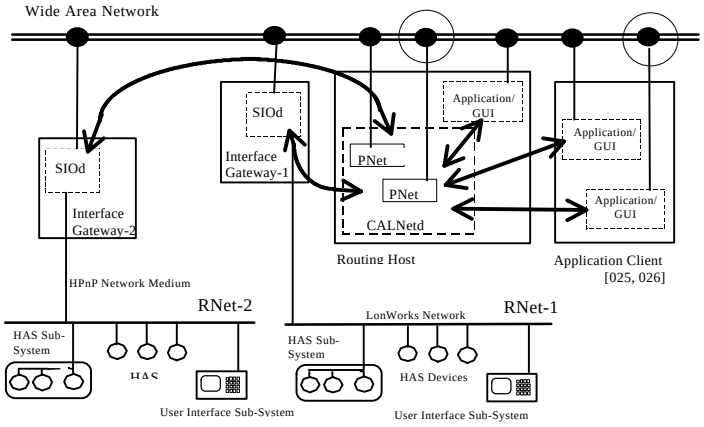

Fig 15: *Different home networks integrated via multiple SIOd daemons, running on a dedicated "modem-like" hardware subsystem*

7. Conclusions

In this paper we have described the various AI techniques and technnologies which allow modern consumer applicances to communicate and interact with one another over a home network. We also have shown how bridging between the home network and an external TCP/IP network, such as the Internet, may be achieved.

Finally we have described how the software architecture of our bridging system is flexible enough to integrate a multiplicity of home networks through a single access point which can provide intelligent brokering to external network services applications programs.

Biographies

Peter Corcoran received the BAI (Electronic Engineering) and BA (Maths) degrees from Trinity College Dublin in 1984. He continued his studies at TCD and was awarded a Ph.D. in Electronic Engineering in 1987 for research work in the field of Dielectric Liquids. In 1986 he was appointed to a lecturship in University College Galway. He is involved in international Joint-Venture projects in both China and Eastern Europe. He is currently teaching on a full-time basis at University College Galway. His current research interests include home automation networks, Internet technologies, embedded computing applications, and telecommunications technologies. He is a member of the IEEE and a technical committee member of the IEEE Consumer Electronics Society.

Dana Vasiloaica received her degree in Computer Science from "Lucian Blaga" University of Sibiu, Romania, in 1997. She is now researching her Masters Degreee in the area of Computer Systems and Networking. Her research interests include ATM and Object Oriented Programming.

REFERENCES

[1] Corcoran P. M. and Desbonnet, J., " A Unified Gateway Architecture to Integrate Home Automation Networks with the Internet" *ICCE '98 - IEEE Conference on Consumer Electronics, Los Angeles, Jun. 1998.*

[2] Hofmann, J., "The Consumer Electronic Bus: An Integrated Multi-Media LAN for the Home", *International Journal of Digital and Analog Communication Systems*, Vol. 4, No. 2, Apr. 1991, pp. 77-86, 1991.

[3] Desbonnet, J., Corcoran P.M., and Lusted, K., "CEBus Network Access via the World-Wide-Web", *IEEE Trans. Consumer Electronics*, Aug. 1996.

[4] Corcoran P. M. and Desbonnet, J., "Web Browser and Applet Interfaces to CEBus Networks", *ICCE '97 - IEEE Conference on Consumer Electronics, Chicago Il*, Jun. 1997.

[5] EIA Home Automation System (CEBus) Standard IS-60, "Common Application Language Specifications", EIA, Part 8, June 29th 1992.

[6] Home Plug and Play Specification, "CAL-based inter-operability for Home Systems", CEBus Industry Council (CIC), April 2nd, 1997; "http://www.cebus.org/hpnp/".