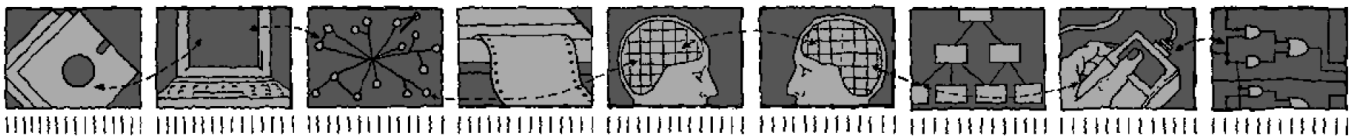
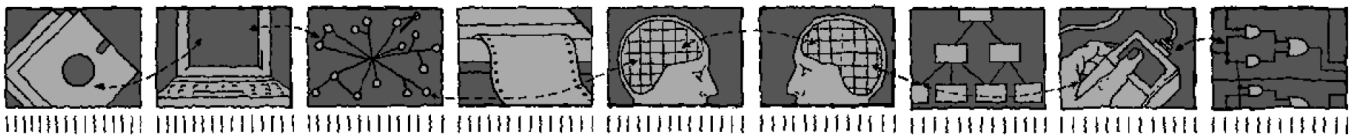


Department of Computing Science and Mathematics
University of Stirling



The Homer Home Automation System

Claire Maternaghan

Technical Report CSM-187

ISSN 1460-9673

December 2010

Department of Computing Science and Mathematics
University of Stirling

The Homer Home Automation System

Claire Maternaghan

Department of Computing Science and Mathematics
University of Stirling
Stirling FK9 4LA, Scotland

Telephone +44 1786 467 421, Facsimile +44 1786 464 551
Email cma@cs.stir.ac.uk

Technical Report CSM-187

ISSN 1460-9673

December 2010

Abstract

This report discusses Homer, a system developed for managing home automation and telecare. The philosophy and architecture of Homer are explained. The nature of home components is discussed, along with how they fit together into the overall system. Policies are used as a means of automated decisions based on user-defined rules for control of the home system. User-friendly interfaces for home management are then presented. Finally, the report summarises what has been covered, evaluates the current status of home automation and telecare, and identifies trends and future developments in these fields.

Contents

1 Introduction

Computing has already pervaded the home through use of personal computers, mobile phones, and microcomputers embedded in domestic appliances. This report discusses how home automation and telecare are bringing a new dimension to pervasive computing in the home. This allows equipment and facilities within the home to be combined and made available through range of technolgies, platforms and interfaces . As a concrete example, the Homer system shows how a common architecture and approach can support both applications.

This section introduces home automation and telecare, and presents the general challenges that they pose.

1.1 Home Automation

The concept of home automation has been around for many years. There are a number of commercial solutions, but these would be better termed home control as they mostly support control of aspects of the home such as lighting, heating, security, audio and video. However, they tend to lack flexibility in combining the control features and hence result in rather rigid and tailored solutions, rather than one flexibile solution that the user can customise to meet their changing needs.

Due to the widespread use of personal computing, mobile phones, media players and the like, consumers have become increasingly knowledgeable about technology, more comfortable with its use, and more acceptive of technology in their every day lifes. The time is therefore ripe to offer more sophisticated ways of managing the home. This can deal with aspects such as comfort, energy efficiency, health, home media and home management.

1.2 Telecare

The global population is ageing, with the percentage of older people (over 65) expected to be 19.3% by 2050 – and much higher in some developed countries [5]. It is both socially desirable for older people to remain in their own homes for as long as possible and infeasible for society to provide sufficient care homes for this growing segment of the population.

Telecare aims to support delivery of care to the home, with a particular emphasis on social care. Telehealth is similar, but focuses on aspects such as health monitoring and support. Remote delivery of care is expensive in manpower, so computer-based support is an attractive and effective solution. Telecare and telehealth can monitor undesirable situations such as night wandering or abnormal medical readings. However, they can also identify potential problems in daily life such as poor sleeping patterns or reduced meal preparation.

Telecare resembles home automation in requiring management of how the home reacts. Although the two applications have some overlap, telecare makes use of specialised devices such as medication monitors, fall detectors and enuresis (bed wetting) sensors. Telecare is usually linked with a call centre for handling alerts. Advantage may also be taken of a wide-area link to upload home care data to a remote facility (e.g. a social work office or a health centre). Not only are telecare systems proprietary, they usually require specialised technical expertise and reprogramming to modify the services they offer.

1.3 Challenges in Pervasive Computing for the Home

Home computing must be appropriate for ordinary householders. Despite increased understanding of computer-based capabilities, consumers will have little understanding of or interest in the technical details of home equipment. The concepts and interfaces therefore need to be readily understood. Home equipment also needs to be acceptable: devices that look out of place in the home are unlikely to be welcome, and devices that need disruptive installations are unlikely to be accepted.

Interoperability remains a challenge. Although a number of standards are available for home automation, these are often proprietary, low-level, and do not guarantee interworking across different commercial solutions. Since telecare is in its infancy, there is little standardisation of telecare equipment interfaces. User-visible interfaces to home equipment are also proprietary and unlikely to be standardised.

Attempts have been made to introduce flexible home automation through user-defined rules or mappings. However, these usually require technical expertise that ordinary householders are unlikely to have

or to learn. Telecare management is highly specialised, being performed by technicians or care workers rather than end users. Nonetheless, it would be desirable to open this up to less technical users (e.g. end users themselves or their family caregivers).

1.4 Home System User Interfaces

Interface design techniques of relevance to home systems include programming by demonstration, tangible programming and visual programming.

Programming by demonstration (e.g. a CAPpella [3], Alfred [4]) allows the user to set up a situation and then demonstrate how the system should respond to it. However, it can take significant effort to demonstrate all the situations that might arise. It can also be difficult to demonstrate rare events.

Tangible programming uses real-world analogues to define system behaviour. Accord [10] defines rules by assembling jigsaw pieces. Media Cubes [1] are similarly used to define rules by placing action requests next to devices. As noted by the designers of Camp [12], approaches like these require users to think in unnatural, device-oriented terms. Instead, Camp focuses on requirements expressed using words from a 'magnetic poetry' set. However, what can be expressed is deliberately restricted to avoid complex natural language processing.

Visual programming is an attractive option for home control. The approach of [6] allows end users to define rules graphically in ubiquitous computing environments, though what may be stated is very restrictive. iCAP [11] allows new devices to be defined by drawing icons. These are then dragged onto a situation window (for rule conditions) or an action window (for system response). Oscar [9] allows components to be interconnected visually, but is almost entirely focused on home media.

1.5 Chapter Overview

The Homer project, discussed in Section 2, offers a generic solution for both home automation and telecare. The philosophy and high-level architecture of the system are described. The types of components are described, and how they are flexibly integrated into the system. Above the components, policies are automated rules for how the home should react to various situations. Finally, user-friendly interfaces at the top level aim to make it easy for non-technical users to manage their homes. Section 6 summarises the chapter, and evaluates the overall state of the art. Future trends in home automation and telecare are identified.

2 Homer System

This section discusses the Homer system, which has been fully implemented unless otherwise stated. The overall architecture, its components, internal framework, policy server and user interfaces are discussed.

The aim was to develop a flexible and dynamic framework that supports any type of device, appliance or home service. The framework focuses on how to represent components in a way that makes them easy to use as building blocks in higher-level services. The framework can then combine these building blocks in a variety of ways to create applications, services, rule-based systems, etc. It is possible to use the same components for completely different purposes, and to reconfigure them dynamically as requirements change. This supports the design principles recommended by Davidoff *et al.* [2] for developing end-user programming systems within a smart home environment.

Considerable device functionality is available within a typical home. Unfortunately, it is not yet common to combine the capabilities of individual devices. Audio-visual systems are the exception, but even there the combination is limited to simple and fixed interconnections. Basic tasks like managing home appliances from a remote location are possible, yet integrated solutions are not readily available. A home system should allow the user to manage and interconnect home devices as required. The Oscar project carried out by Newman *et al.* [9], discussed in [8], demonstrates such a system but it is limited to media devices within the home.

2.1 Homer Philosophy

Homer aims to act as a middle layer between users and components (hardware or software), hiding complexity from both developers and users. Components register themselves with Homer, which then

exposes their functionality to the home user. Instances of components can then be created and placed within a model of the home environment. The user can view and control the state of all component instances. Home management policies can also be written to make use of the defined component instances.

A common limitation of current home automation systems is that the home logic is hidden, so that it cannot be controlled or changed by the user. This can result in the home behaving in ways which the user does not understand and cannot discover without contacting the system installer. It is also common to find that changes to the home logic have to be made by the system installer (at cost). Some commercial systems do allow the user to create rules for the home. However the user interfaces for this are often complex and hard to use, meaning that the average householder is unlikely to attempt any changes. Homer includes an integrated policy server that is made available through an HTTP interface. This makes it easy to create new extensions and interfaces.

2.2 High Level Architecture

Networks are fundamental to both telecare and home automation. They link devices and services in the home, and to services provided in the wide area. Home networks connect a broad range of devices, appliances, sensors and actuators.

There is considerable variety in the protocols used within and to the home. A home platform needs to be as device- and protocol-independent as possible in order to handle this variability. The solution adopted here is to abstract the details by treating devices as providing services. It is then possible to treat everything in the home as a service. This gives a uniform and flexible architecture, and also benefits from the advantages of SOA (Service Oriented Architecture). Fortunately, interfaces (drivers) for many protocols exist in commercial or open-source form. It has therefore been possible in this work to focus on the higher-level framework and services. There are many component frameworks. The attraction of using services as components is that the benefits of SOA can be realised, such as loose coupling and easy combination of components.

A wide range of possible framework options were explored; OSGi (Open Source Gateway Initiative, `www.osgi.org`), using the Knopflerfish (`www.knopflerfish.org`) implementation, was chosen as the final framework tool. A full review of the frameworks explored and the reasons for choosing OSGi can be found in [7].

A high level architectural diagram for Homer is shown in figure 1. The rest of this section explains the various parts of the diagram in detail. Firstly, the Homer Framework and Services are explored. Next, the components are described, then it is shown how developers add devices to Homer which are automatically detected and offered to the user. The Java client user interface is shown for managing these components. The policy server is explained in section 4, and the webserver and external interface are explained in section 2.5. Finaly, external Homer applications are shown and discussed in section 5.

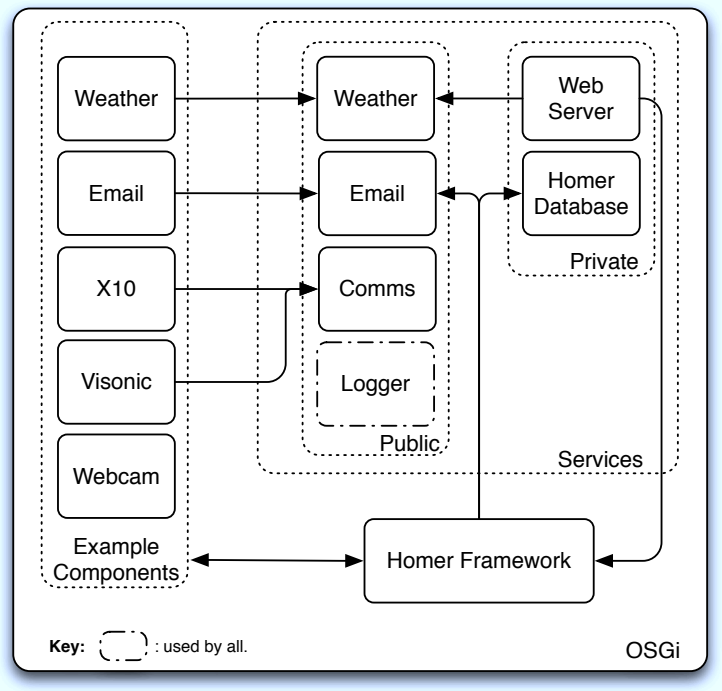


Figure 1: High-level architecture diagram for Homer

2.3 Homer Framework

The Homer Framework bundle, shown in figure 2, is the core of the Homer system. It handles the following apsects:

2.3.1 Component Bridge

The component bridge is used for communication between components and Homer. It extracts relevant information from registered Homer components, as demonstrated above, and sends information to the components such as requests for a particular action to be carried out.

2.3.2 System Bridge

Similarly, the system bridge is used for communication between Homer and private services such as the Homer database and web server.

2.3.3 Policy Server

The policy server is descibed in more detail in section 4.

2.3.4 Event Server

The event server is a hook for a future link to the research carried out on so-called 'device services' [13]. This allows component-level events to be flexibly mapped to/from policy-level events through external logic defined by web service orchestration. This supports programmable sensor and actuator fusion. It is planned to investigate future use of this by Homer.

2.3.5 Event Hub

The events hub acts as the central communicator for all interested parties. For example, the component bridge will listen for any condition checks or action requests for each system device type, so as to then

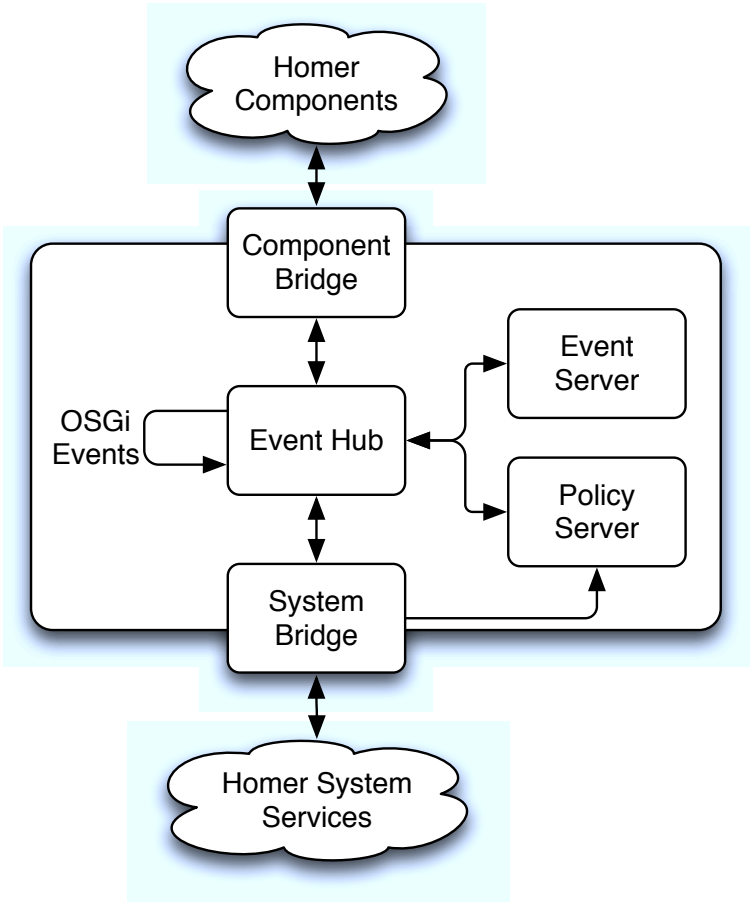


Figure 2: Homer Internal Architecture

contact the relevant component. The component bridge would also tell the event hub about any triggers from its components, as means of distributing such messages.

Homer makes use of OSGi's event messaging service where events can be posted and listened for. Events carry the IDs for various pieces of information about the particular device. This includes system device information (e.g. type 'X10 Appliance Module', instance 'B3' as X10 module address), user device information (e.g. type 'lamp', instance 'bedside light'), location information (e.g. context 'home', location 'bedroom'), and event details (e.g. trigger 'turned on', condition 'is off', action 'dim' with parameter '50'). This allows different system elements to listen for different kinds of information. For example, a Twitter service could listen for any kind of trigger event and report this via a tweet. As another example, a kitchen display could listen for events in the kitchen to keep the device statuses up-to-date.

The format of Homer OSGi events is:"uk/ac/stir/cs/homer/<system device type id>', with the following list of attached parameters: "UserDeviceID", "UserDeviceTypeID", "SystemDeviceID", "LocationID", "LocationContextID", "EventID", "EventType", and optionally: "Parameters".

2.3.6 Data Type Specification

A protocol for defining data types that can be translated to JSON (JavaScript Object Notation, `www.json.org`) for sending to interested systems such as a webpage or iPad application. These data types are to allow component developers to define the type of information they may need. This was seen in the examples above when a new X10 Lamp or Appliance module was created: the user must be asked for the 'house code' which is a string of 2-3 characters, and also when the user sets the light level for a particular device which uses the X10 Lamp Module the user must be asked for a 'percentage' which is a number value between 0 and 100. This information can be parsed before being displayed to users in any way seen fit by the user interface developer/designer.

2.4 Homer Services

Bundled with the Homer Framework Home System are simple services which offer helpful tasks for developers. Some of these services are 'private', meaning that they cannot be used by components, while others are public, meaning that both the internal system and the components can freely make use of them. Currently the services available are:

2.4.1 Logger

A simple logging bundle which offers the Logback (`logback.qos.ch`) API to other bundles. It also ports Java Util Loggings and Knopflerfish (the implemetionation of OSGi Homer uses) to Logback automatically too.

2.4.2 Comms

Provides the functionality for components to be able to connect, and communicate with, an RS232 or USB port.

2.4.3 Weather

A basic service which uses the Google Weather API to obtain the current weather for a particular location.

2.4.4 Homer Database

The Homer Database uses the H2 Database Engine (`www.h2database.com`) to store and access all the data involved within the home system. H2 is an open-source Java SQL database and was chosen as it is a popular and fast, with a small footprint. This includes all the system device types and system devices, the user device types and user devices, the possible triggers, conditions and actions, and any policies. The Homer Database API provides a clean external access point to the data, which is available for the Homer Framework and other internal bundles such as the Policy Server and Web Server.

2.4.5 Web Server

The Web server bundle makes use of the inbuilt OSGi HTTP Web Server to offer external access to Homer. This external interfaces is described in the following section.

2.5 External Interface

The web server bundle of Homer exposes an HTTP-based API that supports HTTP requests for information. Each HTTP request requires an application key and secret key as a means of simple authentication before any information is returned. The full functionality of the system is available through the server allowing any external technology to make use of Homer. Such examples could include: iOS (for iPhone and iPad development), web technologies for a web client, Android (for Android mobile phone) applications and Flash applications.

The Homer HTTP API uses JSON as a means of sending and receiving chunks of infomation as it is a simple and well-supported format. As an example, an application could be made that allows the user to control the home, so first of all a list of the location contexts that there could be devices within is shown to the user. So it could display "Home" and "Work", as both of these have devices that could be controlled, however "Neighbour's House" is not shown since we cannot do anything to their house, we can only query some information about that location. The following JSON would be sent to Homer:

```
queryObj: { operation:"getLocationContexts" actionID:"*' }
```

Homer would then look up all the location contexts which have any type of action and return to the external application:

```
result:
{
   [
      "id": "<some string id, e.g. '1'>"
      "name": "<some string name, e.g. work">
      "image": "<some string image id which you can use to request an image
             from homer (all images are stored within the database, so knowing
             the id of the image allows the client to request the image from Homer)>"
   ]
}
```

This JSON represents an array of location contexts, giving a unique ID, a display name and an image ID for each location context. Assume that the user chooses a location context (say, with ID '1'). The application then needs to display all the locations within the chosen location context which contains any actions. The external application would send:

```
queryObj: {operation:"getLocations" locationContextID:"1" actionID:"*" }
```

Homer would return an array of locations within location context '1' that have one or more deviceswithin that have actions:

```
result:
{
   [
      "id": "<some int id, e.g. 1>"
      "name": "<some string name, e.g. office">
      "contextID": "1" (the location context that the location is in)
      "image": "<some string image id which you can use to request an image from homer>"
   ]
}
```

This JSON respresents an array of locations, giving a unique ID, a display name, the location context it resides in and an image ID.

2.6 User Oriented Features

Homer is being developed iteratively, both device-up and user-down. This means that features are continually added to Homer in order to support the user interface ideas. These features are discussed in Section 5 from the user point of view, rather than the system point of view. Some examples of these features are sensor and actuator fusion (saving 'when' and 'do' clauses for reuse), and the notion of people within the system having associated triggers, conditions and actions.

3 Components

A component within Homer simply wraps and exposes a particular type of device or software service. Each component is implemented as an OSGi bundle, which allows components to be added at run time. As examples of components, there is support for X10 devices (mains-controlled), Tunstall devices (telecare), SMS (messaging) and weather forecasting. This section explores the creation of components from the developers point of view, and the nature of these components to the end-user.

3.1 Introduction

A component within Homer offers some kind of capability to the user. This functionality is associated with a particular system device type, and is split into three different aspects: triggers (events), conditions (states) and actions. A component acts as a proxy between Homer and the underlying hardware or software service. Components are categorised according to their underlying technology. However, users view the system using categories that *they* create and which make sense to them.

As an example, X10 supports a variety of hardware that uses a common protocol and is therefore one Homer component. X10 devices are either appliance modules (on, off) or lamp modules (on, off, dim), which are treated as two different system device types. Users have access to a simple device management application, shown in section , which allows them to create new instances (user devices) of hardware within the home. Users create their own categories (user device types) instead of having to refer to the system device type names (here, X10 appliance modules or X10 lamp modules). This allows users to name and group devices in ways meaningful to them, without having to be aware of the underlying technology. For example, user-defined categories for X10 devices might include heating and lighting.

As a further example, Visonic make a range of wireless sensors for the home. Being supported by a common protocol, these are managed by one Homer component. The different kinds of sensors (movement, opening, etc.) are treated as different system device types. As the user installs these sensors, they can be allocated to categories meaningful to the user (security, doors, etc.). Decoupling the user view of devices from the underlying technology makes the system more usable. Users can refer to devices within the home as they wish. Devices are associated with particular technologies only at installation time.

All components offer services in the form of triggers, conditions and actions; these are specific to each system device type. For example an X10 appliance module system has actions 'turn on' and 'turn off', whereas an X10 lamp module also supports 'dim'. Components report their advertised triggers, check their advertised conditions on request, and perform their advertised actions.

Multiple underlying technologies can support one user device. For example, the lounge TV might be controlled by both an X10 appliance module for on/off as well as an infrared controller for changing channels and volume. For any user device, the actions that can be carried out are those of the parent system device types. Suppose the kitchen TV made use of only an X10 appliance module. Both TVs could be powered on or off, while the lounge TV could also have its channels and volume changed.

Device support has been created for use in both home automation and telecare. Examples in various categories which Homer currently supports are as follows:

- Appliance Control: Appliances controlled via the mains include lighting, fans and TVs. Appliances controlled via infrared include TVs, audio-visual systems and DVD recorders.
- Communication: Communications services include email, SMS (Short Message Service), Facebook, Twitter, message display on a digital photo frame, and speech input/output (using code from the University of Edinburgh).

- Energy Consumption: Energy usage will be monitored per appliance once suitable hardware becomes generally available for the UK. This will allow Homer to react to how much energy is being used and help reduce energy consumption. For example, clothes washing might be delayed until other energy demands are lower.

- Environment: Oregon Scientific sensors (`www.oregonscientific.com`) are used for humidity, temperature, etc. The Google Weather API is used to obtain the current weather or a forecast for chosen locations.

- Home Automation: Sensors from companies like Tunstall (`www.tunstallhealth.com`) and Visonic (`www.visonic.com`) include movement detectors, pressure mats and reed switches (cupboard, door, window). Homer is capable of handling a variety of home actuators (though the current support is limited). Future support will include curtain/blind controllers, garage door controllers, and remote door locking and unlocking.

- Telecare: Telecare sensors from Tunstall and Visonic include alarms (pendant, wrist), hazard detectors (flood, gas, smoke), medicine dispensers, and pressure mats (bed, chair). Specialised sensors include detectors for enuresis, epileptic seizures and falls.

- User Interfaces: Various 'Internet buddies' are supported as they appeal to ordinary users. Examples of these are the i-Buddy 'angel' (`www.unioncreations.com`), the Nabaztag 'rabbit' (`www.nabaztag.com`), and the Tux Droid 'penguin' (`www.ksyoh.com`). The WiiMote (`www.nintendo.com`) can be used to communicate using gestures and tactile output. Using code from the University of Glasgow, similar functions are available from the Shake (Sensing Hardware Accessory for Kinaesthetic Expression, `www.dcs.gla.ac.uk/research/shake`). The Homer interface described in Section 5 supports iPad (`www.apple.com/ipad`), iPhone (`www.apple.com/iphone`) and web interfaces. These offer different services and means of control to the home user.

3.2 Developing Components

Each component or service is an OSGi bundle which must import the HomerFrameworkAPI bundle. An OSGi bundle activator (the main class for an OSGi bundle) simply registers its component class with Homer. The component class must implement HomerComponent, which has required methods to obtain a list of hardware types (for example X10 Appliance Module and X10 Lamp Module) or high-level system services (for example Weather or SMS). Other methods register, edit and delete a component instance (for example edit the bedroom lamp's X10 address from A1 to A2 in the X10 bundle).

The component must also implement WhichHasTriggers, WhichHasConditions and/or WhichHasActions dependent on the functionality offered by the system device types supported by the component. This provides Homer with access to necesary methods to check a condition is true or false (for example, is the bedroom lamp on?), request that an action is performed (for example, turn on the bedroom lamp), or to get a list of triggers, conditions and/or actions for each hardware type supported by that bundle. These methods allow Homer to offer the features of the component to the user.

An example of a Homer component bundle could be X10. When the X10 bundle starts:

- It registers itself with Homer:

  ```
  ComponentGateway.Singleton.get().registerComponent(new X10Component());
  ```

- Homer then queries the X10 component to find out what system device types the X10 bundle supports (described above as a hardware type or high level system service).

- The X10 component returns a unique identifier and initial user-friendly name for both an X10 Lamp Module and an X10 Device Module. Homer offers a hashing function to produce a unique string identifier when given a class type and string name. Sample code shown in code 1.

These IDs will be consistent each time the bundle starts and therefore can be used reliably within the X10 bundle to refer to each different type of device supported and also throughout the rest of Homer.

```
static final String X10_APPLIANCE_MODULE_SYSTEM_TYPE =
              IDUtil.getUniqueIdentifier(X10Component.class, "X10 Appliance Module");

static final String X10_LAMP_MODULE_SYSTEM_TYPE =
               IDUtil.getUniqueIdentifier(X10Component.class, "X10 Lamp Module");
```

Code 1: Creating Unique System Device Type Identifiers

As well as returning an ID and initial user-friendly name the bundle must also return any information required for instantiating an instance of the device type. Creating a new device which has the hardware of either an appliance module or a lamp module requires the X10 house code that is associated with the device.

The method to obtain the list of supported system device types is shown in code 2.

```
public SystemDeviceType[] getSystemDeviceTypeData()
{
   // create the text field which is needed for the user to input the house code
   // "A1" is the default text, minimum of 2 characters, maximum 3, and has no units.
   HomerText houseCodeTextField = new HomerText("A1", 2, 3, Unit.NONE);

   // create a configuration part which for the text field
   // "X10 Code:" must introduce the text field on screen and
   // there is no need for text after the text field.
   ConfigPart configPart = new ConfigPart("X10 Code:", houseCodeTextField, null);

   // create the configuration data object to store our text field
   // it can hold any number of configParts; in this case we only have one text field.
   ConfigData configData = new ConfigData(configPart);

   // return the list of supported system device types,
   // with unique identifier, initial name and any configuration data.
   return new SystemDeviceType[]
      {
         new SystemDeviceType(X10_LAMP_MODULE_SYSTEM_TYPE,
                              "X10 Lamp Module", configData),
         new SystemDeviceType(X10_APPLIANCE_MODULE_SYSTEM_TYPE,
                         "X10 Appliance Module", configData)
      };
}
```

Code 2: Get System Device Type Data

- Homer then looks up the Homer database to see if it already knows about these device types:
  - If it does not, then add this device type to the database.
  - If it does, then check to see if there are any instances (system devices) of that system device type in the database:
    - if there are no system devices then do nothing.
    - if there are system devices which are of system device type X10 Appliance Module or Lamp Module then let the X10 bundle know about each by calling the `registerComponentInstance` method on the X10 component, giving it the unique identifier for the system device along with a string array of data that is necessary for the system device type. In this case this is an array with one string in it which would be the X10 house code. The X10 component should store the house codes (along with the associated system device ID) for being asked

at a later time to carry out some action on or check about a particular system device. This saves the component having to look up information in the database or in Homer.

- Homer will also check the component for triggers, conditions and actions (checking every time the bundle is manually restarted in case the bundle was updated) for each system device type supported by the bundle, and register these in the database. The information required for each is a unique identifier (generated using the same technique described for system device type), string friendly name and a default icon. In the X10 case we would return the same set of triggers, conditions and actions for both appliance and lamp module, except with the additional action of 'dim' for the lamp module. The code for getting the actions is shown in code 3.

```
public List<Action> getActions(String deviceTypeID) {
   List<Action> actions = new ArrayList<Action>();

   if (X10_APPLIANCE_MODULE_SYSTEM_TYPE.equals(deviceTypeID))
   {
     actions.add(new Action(AM_TURN_ON_ID, "turn on", Action.DEFAULT_TURN_ON_IMAGE));
     actions.add(new Action(AM_TURN_OFF_ID,"turn off", Action.DEFAULT_TURN_OFF_IMAGE));
   }
   else if (X10_LAMP_MODULE_SYSTEM_TYPE.equals(deviceTypeID))
   {
     actions.add(new Action(LM_TURN_ON_ID, "turn on", Action.DEFAULT_TURN_ON_IMAGE));
     actions.add(new Action(LM_TURN_OFF_ID,"turn off", Action.DEFAULT_TURN_OFF_IMAGE));

     // when dimming the light the desired dim percent is required
     // it is a number, range from 0 to 100, increments of 10,
     // 0 decimal places, default value of 50, % unit
     Parameter lightPercentParam = new Parameter(new HomerNumber(0, 100, 10, 0, 50,
                                                                 Unit.PERCENT));

     // Parameter data is created from a collection of parameters
     // in this case it only needs one parameter
     ParameterData paramData = new ParameterData(lightPercentParam);
     actions.add(new Action(LM_DIM_ID, "set light level",
                            Action.DEFAULT_TURN_ON_IMAGE, paramData));
   }
   return actions;
}
```

Code 3: Get Actions

3.3 Managing Components

Creating new instances of components is not something that is done at system level. Instead a simple user interface has been created to demonstrate the separation of this logic and ease for the user. This user interface is a simple Swing application: a separate OSGi bundle which can be started along with the other OSGi bundles (Homer framwork, components, etc.) or ommited from this set of start-up bundles. It is purely to demonstrate the principles and would, in reality, be made a lot more pretty and accessible (most likely from the iPad application described in section 5 or web-based interface).

So far the terms System Device Type and System Device have been introduced. These terms refer to the hardware or a fundamental service. They are usually hidden from the user and only exposed when hardware changes need to be made. An entity relationship diagram is shown in figure 3 to demonstrate the relationships between the various terms.

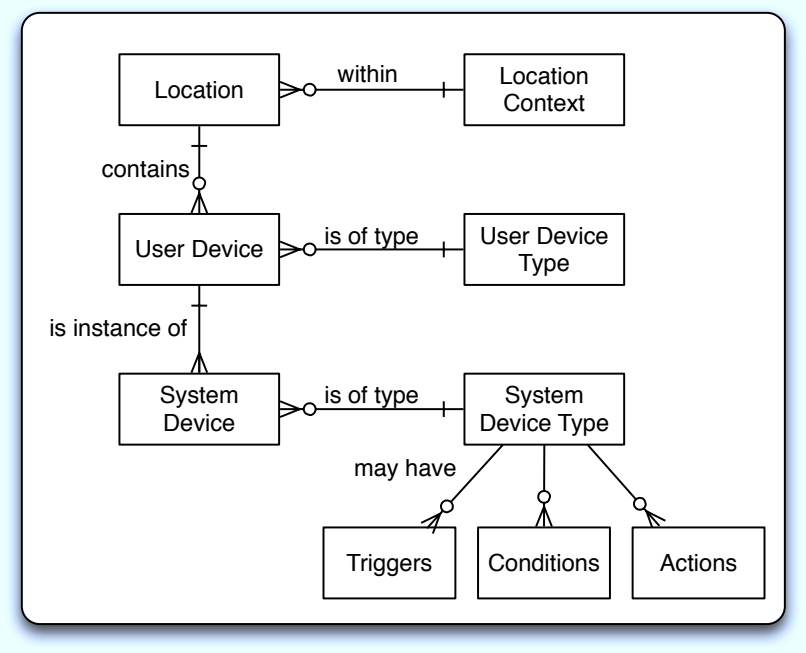


Figure 3: Entity Relationship Diagram of the Homer Terms

The screenshot in figure 4 shows the hardware management screen displaying two hardware types: X10 lamp module and X10 appliance module. These are the system device types. There are two instances of each system device type: reading lamp, bedside lamp, fan and TV. These are the system devices. This view can add, edit and remove system devices. The system device types are from what the registered bundles offer, as seen in the code examples above. The only aspect of this which can be changed by users is the name for the system device type (in case there is a name they would find it easier to refer to certain hardware).

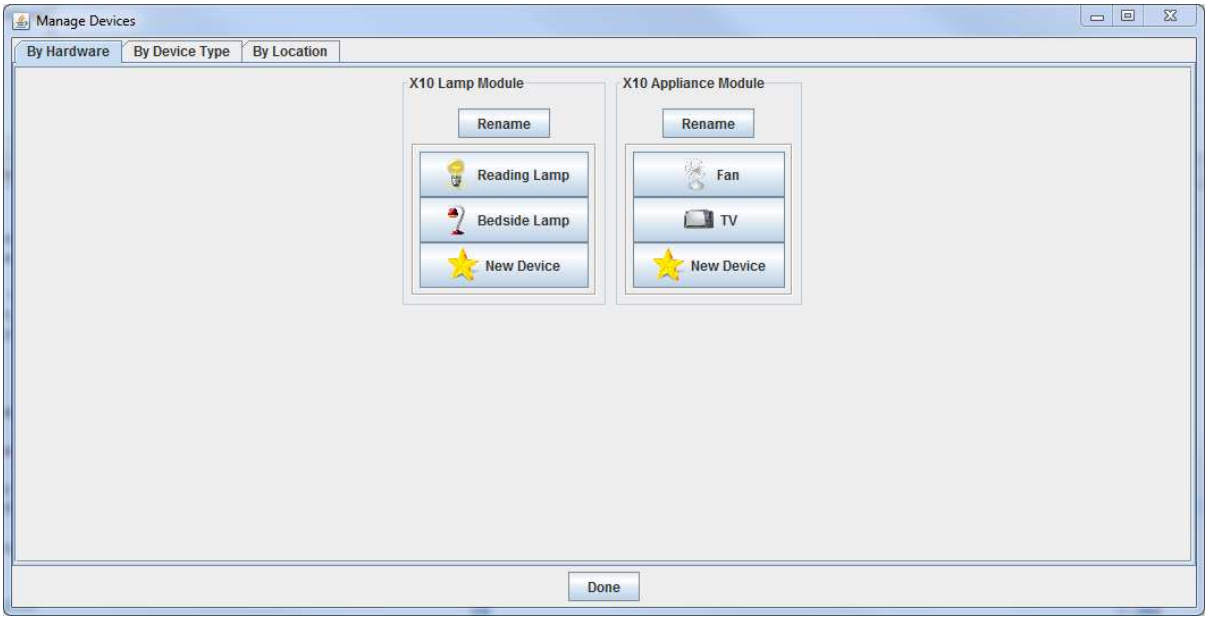


Figure 4: Hardware Management Screen - by System Device Type

System device types are correlated directly to the hardware. This needs a different, more user focused,

means of referring to devices within the home. This has resulted in the concept of user device types and user devices. User device types are user-friendly names for types of devices within the home which are chosen and managed by the user. The screenshot in figure 5 shows an alternative view of managing devices within the home, this time focused on user device types. In the screenshot there are four different user device types that have been created: lamp, TV, kitchen appliance and fan. There are two lamp user devices, one TV user device and one fan user device. Each user device maps directly to a system device. So, the reading lamp and bedside lamp are both X10 lamp modules that are automatically mapped to a system device of type X10 lamp module. Similarly the TV and fan map to a system device of type X10 appliance module. This allows a user device, say a television, to be supported by more than one system device type, e.g. an X10 appliance module to turn the mains power on and off and an infrared controller to control the television using infrared.

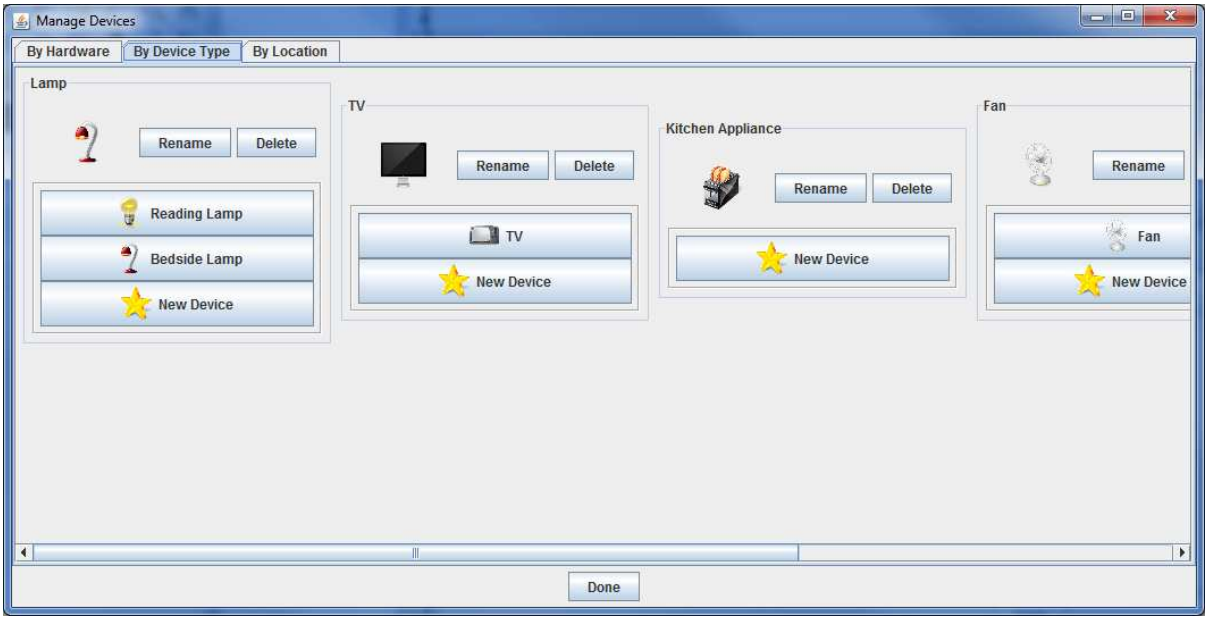


Figure 5: Homer Management Screen - by User Device Type

A property of a user device is its location, and a location must have a location context. The screenshot shown in figure 6 shows this concept by representing the devices in the home in yet another view. In this screenshot the location context shown is Home, with the locations Living Room, Bedroom and Utility Room. From this view we are able to add, edit and delete location contexts and locations, as well add new devices to particular locations. We can see also that currently our Reading Lamp, Fan and TV are all located within the living room and the bedside lamp within the bedroom. The utility room does not contain any devices at the moment.

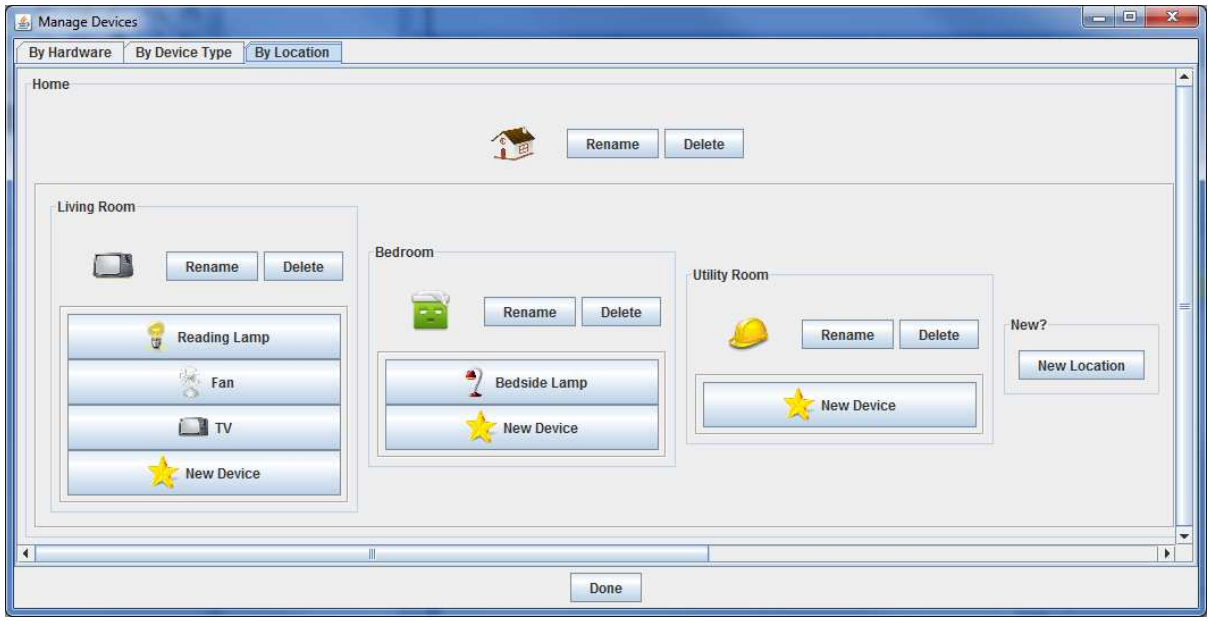


Figure 6: Homer Management Screen - by Location

When adding or editing a device from any of the views shown in figures 4 to 6, the user is presented with a screen shown in figure 7. This form allows the user to choose an appropriate image for their device, a name, the system types, the user type and finally any properties that are required by any of the system types that the user chooses. For the fan in figure 7, which is of hardware type X10 Appliance Module, the form shows a text field at the bottom of the form asking the user to enter the house code for the fan. This text box is shown as the developer of the X10 appliance module requested that it needed a string parameter (demonstrated in the code samples above).

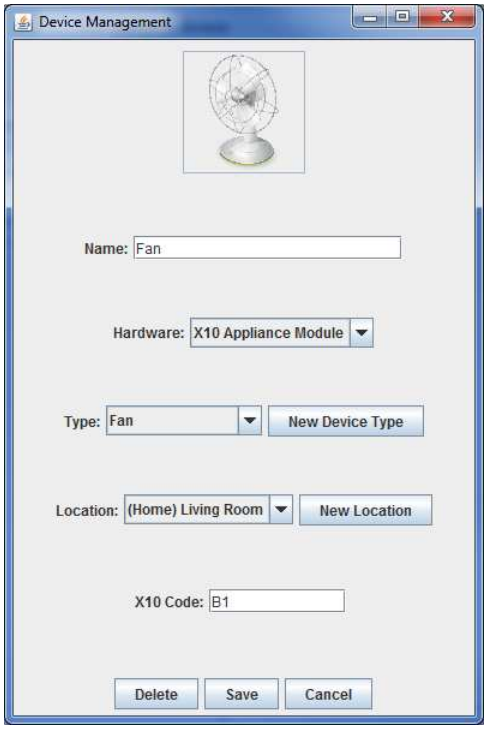


Figure 7: Homer Screen shot: adding/amending/deleting a user device.

3.4 Controlling and Automating Components

Once a component bundle has started, Homer requests information about the component. As mentioned above, it requests information about the types of device types it offers for allowing the system to offer users the functionality of adding new devices of that device type. At this same stage Homer also asks the component for any triggers, conditions and/or actions (TCAs) that each of the registered system device types supports (as shown in the code example in section 3.2). All TCAs are registered in the Homer database and associated with the particular system device type. This means that when a user adds a new user device, say a Bedroom Lamp of user device type Lamp, Homer will use the system device type X10 Lamp Module. Homer will also know that the bedroom lamp can be turned off, turned on or set to a specific light level. Another example could be a television. Suppose the user defined a device Main TV of type TV which used the system device types X10 Appliance Module and InfraRed. Homer would then amalgamate the actions for X10 Appliance Module and Infrared to conclude that it could turn the TV on, turn the TV off, change the channel and volume, and any other features that the InfraRed component for that particular TV could offer. Similarly there could be a Kitchen TV that is also of the user device type TV but only supports the X10 Appliance Module system device type; in this case, Homer could only turn the TV on or off.

All registered triggers, conditions and actions for each system device type can have their textual description and associated images edited by users if they so desire. The screenshot in figure 8 demonstrates this. Because all the TCAs are referenced by their unique identifiers the user can change the names without having any impact on the system or saved policies.

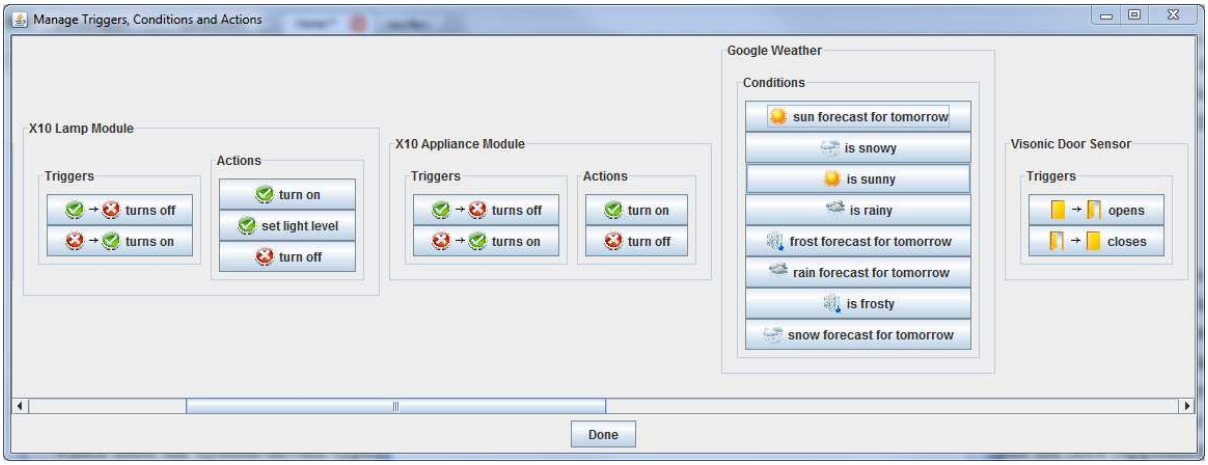


Figure 8: Editing triggers, conditions and actions

Components can be included automatically in policy definitions. For example, the X10 Appliance Module supports the triggers 'turned off' and 'turned on' and the actions 'turn on' and 'turn off'. For a fan, of type Fan, called Fan which supported the X10 Appliance Module then when defining a policy the user will be offered these functions. So the user could put together a policy which said: 'When Fan turns off then turn on fan'. Although this is not a sensible policy, it demonstrates how the information we know about the Fan can be put together to create policies. For more on policies see section 4.

4 Home Policies

Policies define how the home should react to events. A wide variety of policies can be defined for both home automation and telecare. These policies cover aspects such as appliance control (e.g. lighting control), communication (e.g. how to be contacted), comfort (e.g. room temperature), entertainment (e.g. favourite programmes), modalities (e.g. use of speech), reminders (e.g. appointments), security (e.g. intruder detection), system aspects (e.g. access control), and telecare (e.g. medication alerts).

4.1 Format

Homer policies have a 'when-do' format. The **when** clause comprises triggers and conditions that can be combined with **and**, **or** and **then**. The **do** clause can contain multiple actions and conditional groups. A policy is represented as a tree: a hierarchy of terms combined with explicit precedence. How a particular user interface implementation decides to display this to the user is irrelevant to Homer (see Section 5). The following examples illustrate various policy formats:

> When the house is unoccupied and it is dark do turn on the hall light.
>
> When John arrives home or Mary arrives home do play music and (if the outside temperature < 10 then do turn on the heating).
>
> When someone gets up at night then the front door is opened do say 'go back to bed' and illuminate a path back to the bedroom.

Users can find it difficult to differentiate between triggers (e.g. the door opens) and conditions (e.g. the door is open). This confusion is eliminated by treating triggers and conditions similarly. For each element in a **when** clause, the policy server listens for triggers as well as state changes that affect conditions. State changes imply that something has happened to trigger the change of state, allowing the rest of the **when** clause to be evaluated. Conditions are also evaluated when an event triggers a policy.

It is very rare for triggers to occur at the exact same time, therefore policies have a time interval in which all triggers occur and all conditions are met. The time interval depends on the particular policy being defined. For example, take the following two **when** clauses: the first trying to determine if the house owner has arrived home and the second determining if a visitor arrived at the home:

> When the garage door opens then the garage door closes then the front door opens (within 5 minutes)
>
> When movement is detected on the porch and the front door bell is pressed (within 1 minute)

4.2 Language

The Homer Policy language is expressed in code 4.

4.3 Representation

A policy is represented as a tree structure, wrapped using JSON. When Homer is given a new policy it is saved to a database and passed to the policy server. This then loads the policy into a custom tree structure. The **when** part of the tree is represented with nodes of one of the following types: **and**, **or**, **then** or **event**. Each node stores any child nodes or, in the case of an event node, stores the details of the particular trigger or condition. The **do** part of the tree is, again, represented as a tree. The only nodes allowed within the **do** part of the policy are **event** and conditional (**if**) nodes. The same principles apply as for the **when** part, with hierarchies of terms with **action** end nodes.The explicit representation in JSON is shown in code 5.

As an example policy which would read to the user:

> **When** Office Door opens **then** Movement is detected **then** Office Door closes **do** turn on the desk light.

Written in JSON in code 6.

4.4 Implementation

A policy can be enabled or disabled by the user; this dictates whether the policy should be listening for events. Multiple policies can exist at run time. Currently the policy server has been tested with 50 enabled policies, and is expected to scale comfortably to 500 policies or more. Any more than that could be argued unlikely within one home. If a policy is enabled then it can be either waiting or activated. A waiting policy is simply waiting for any of its child (or sub-childs *etc.*) nodes to report that an event has happened. If a policy is activated, one of its child nodes has reported that something has happened. The

```
policy: scenario procedure;

scenario: "when" scenario_node ("within" duration)?;
scenario_node: scenario_and | scenario_or | scenario_then | scenario_event;
scenario_and: "and" '{' scenario_node+ '}';
scenario_or: "or" '{' scenario_node+ '}';
scenario_then: "then" '{' scenario_node+ '}';
scenario_event: "event" user_device_id event_id type parameters?;

duration: integer;
integer: ('0'..'9')+;

user_device_id: quoted_string;
event_id: quoted_string;
type: "trigger" | "condition";
parameters: "params" '{' quoted_string+ '}';

quoted_string: '"' character '"';
character: /* any Unicode character except '"', which is escaped as '\"' */ ;


procedure: "do" procedure_node;

procedure_node: procedure_and | procedure_if | procedure_action;
procedure_and: "and" "{" procedure_node+ "}" ;
procedure_if: "if" condition '{" procedure_node+ '}' ("else" "{" procedure_node+ '}')?;
procedure_action: "action" user_device_id event_id parameters?;

condition: user_device_id event_id parameters?;
```

Code 4: Policy JSON Structure

```
policy: {
   "id": "unique id generated by homer and used for unique identification"
   "name": "user chosen name for the policy"
   "author": "name of who wrote the policy"
   "dateCreated": time in millis of date created date
   "dateLastEdited": time in millis of the date the policy was last edited
   "enabled": boolean flag stating if the policy is currenly enabled
   "timeInterval": number of seconds that the when clause's triggers must occur
                   and/or conditions must be met
   "when": {
      "AND/OR/THEN": [Ordered list of children, with the same options as that
                      of the children of "when"]

      OR an event node:

      "EVENT": {
         "userdeviceid":"unique id for the user device"
         "eventid": "unique trigger or condition id"
         "type": "TRIGGER" OR "CONDITION"
         (optional) "parameters": [Ordered list of string values]
      }
   }
   "do": {
      "AND": [Ordered list of children, with the same options as that
              of the children of "do"]

      OR an if node

      "IF": {
         "CONDITION": {
            "userdeviceid": "unique id for the user device"
            "eventid": "unique condition id"
            (optional) "parameters": [Ordered list of string values]
         }
         "IF_TRUE": [Ordered list of children, with the same options as
                     that of the children of "do"]
         "ELSE": [Ordered list of children, with the same options as that
                 of the children of "do"]
      }

     OR an event node:

      "EVENT": {
         "userdeviceid": "unique id for the user device"
         "eventid": "unique action id"
         (optional) "parameters": [Ordered list of string values]
      }
    }
}
```

Code 5: Policy JSON Structure

```
policy: {
   "id": "123"
   "name": "Turn on light when someone walks into office"
   "author": "Claire"
   "dateCreated": 123456
   "dateLastEdited": 123789
   "enabled": true
   "timeInterval": 60
   "when": {
      "THEN": [
         {"EVENT": {
             "userdeviceid":" 24"
             "eventid": "u185DDD121346C5A207A4536DE7ABC707E9186C18"
             "type": "TRIGGER"}
         },
         {"EVENT": {
             "userdeviceid":" 18"
             "eventid": "1F960252B3EF9E51B2580148246090F990BA3EA2"
             "type": "TRIGGER"}
         },
         {"EVENT": {
             "userdeviceid":" 24"
             "eventid": "76E4D7FE437D04A8B323EEC25427113AB911001F"
             "type": "TRIGGER"}
         }
      ]
   }
   "do": {
      EVENT": {
         "userdeviceid": "25"
         "eventid": "7719316A8D71BE5EEBCD0B2E78B4702BAA696781"
      }
    }
}
```

Code 6: Policy JSON Example

policy therefore starts a countdown according to its time interval (the default time being 60 seconds). As events occur they notify their parent node. As a node becomes true it notifies its parent. If the root node becomes true, it tells the policy which executes and resets the child nodes to false. If the time interval passes before the **when** is satisfied then the policy does not run and its child nodes are reset.

In the case of an **and** or **or** node, a listener is registered for each child node, whereas a **then** node registers a listener only for its first child. If an **or** node has a child whose trigger/condition becomes true, its child nodes will stop listening until the policy is reset. If a child of a **then** becomes satisfied, it will stop listening for that child and instead start listening for the next child. By listening only for what is relevant, the policy server can run efficiently and reduce the number of event listeners required.

4.5 Sample Policies

This section will give some sample policies that could be expressed in the Homer policy language:

Lighting

- **When** front gates detect a vehicle entering **do** turn on driveway lights **and** turn on front door lights
- **When** front door is unlocked **and** hall is dark **do** turn on hall lights
- **When** living room is occupied **and** living room light level falls below 60% **do** turn on the side lamp
- **When** time is 10:30pm **and** Mary has work the next day **and** Mary is not in bed **do** dim the living room lights to 50%
  - includes two scenarios:
    - Maryhas work the next day, which is defined by: **when** day is Sunday – Thursday **and** Mary's calendar event tomorrow is 'work'.
    - Maryis not in bed, which is defined by: **when** left-side of bed is unoccupied.

Heating

- **When** Mary is getting up **or** Mary is going to bed **do** keep temperature in bedroom to comfortable house temperature
  - includes two scenarios:
    - Mary is getting up, which is defined by: **when** time is between (time is one hour before Mary's calendar's first event of the day) and (time is Mary's calendar's first event of the day).
    - Mary is going to bed, which is defined by: **when** (Mary has work the next day **and** time is between 21:30 and 2230) **or** (Mary does not have work the next day **and** time is between 22:30 and 2330).
- **When** movement detected in a room **do** keep temperature in room to comfortable house temperature

Music

- **When** music is playing anywhere **and** telephone rings **do** reduce music volume by 80%

Security/Safety

- **When** fire alarm is activated **and** no one is home **do** send SMS to Claire saying 'The fire alarm was activated but no one is home! Calling your next door neighbour.' **and** call next door neighbour saying 'Please help, the Smith's home's fire alarm was activated but they are not home!'
  - includes one scenario:
    - No one is home, which is defined by: **when** no movement detected for more than 1 hour **and** all beds are unoccupied.
- **When** movement is detected outside **and** time is between 2300 and 0600 and house in sleep mode **do** turn on outside light **and** turn on outside security camera

Kitchen

- **When** left side of bed becomes unoccupied and time is after 0630 **do** turn on the coffee machine
- **When** SMS received from Mary saying 'turn on the oven' **do** turn on the oven

Bedroom

- **When** the television in the living room is turned off **and** time is after 9:30pm **do** turn on the electric blanket in master bedroom **and** send SMS to Mary saying 'Electric blanket has been turned on!'
- **When** time is between 0500 and 1200 **and** Mary is in bed **and** diary has an event in an hour **do** activate Mary's alarm clock

Living Room

- **When** living room unoccupied for 5 minutes **do** turn off the television and speakers.

5 Home Interface

The home system needs to be made usable by the home occupant. There are two main parts that need accounted for: the first being the management of the home and its devices (how do users access and control their home?), the second being the creation/configuration/management of home rules and policies (how do users automate tasks, then manage and reconfigure such tasks, within their home?).

Homer provides an HTTP based API that allows external entities to manage the home system. This supports any application that can make HTTP calls with JSON objects. For security, authentication uses an application key and a secret key. This openness makes it easy to write new applications for the home, using different styles and technical approaches. For example, web-based, iPhone and iPad applications have been written.

The web-based application uses the Google Web Toolkit (`http://code.google.com/webtoolkit`) to expose devices activities within the home, browsable by location or device type.

The iPhone application shown in Figure 9 makes it possible to browse devices by location or type. The current state of a device can then be viewed, its history of past events is available, and the device can be asked to perform selected actions. A live Twitter feed is also available for all events within the home.

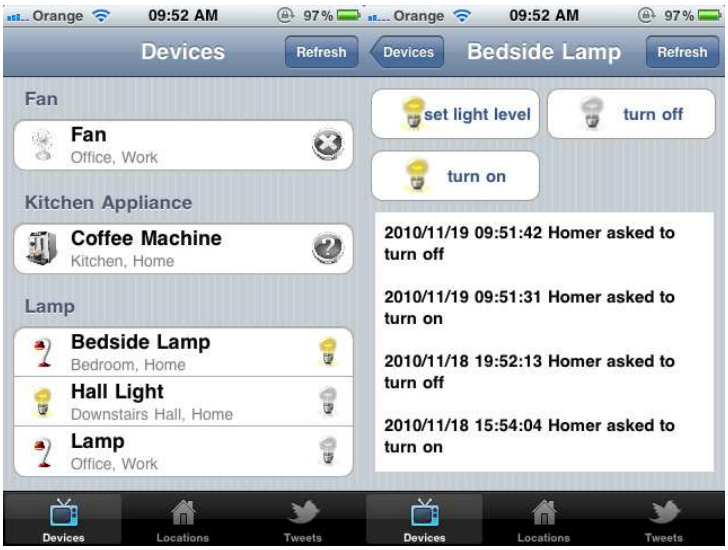


Figure 9: iPhone Application for Homer (two screenshots)

5.1 iPad for Homer

iPad for Homer allows users not only to create and manage the rules within the home, but also to control and manage the home. Generally, end-users feel comfortable controlling the home, whereas they tend to fear the prospect of programming it. They desire the ability to have such rules that can be easily modified and adapted, however they do not like the idea of having to program them. Given what is available with other home automation packages I do not blame them! iPad for Homer brings together controlling and programming the home in one sleek and simple interface. By blending the programming aspects into the control aspects the user should be less likely to both notice, or fear, the programming parts. The other positive of bringing these aspects together is that the user does not necessarily have to make a conscious decision to program the home by opening a new piece of software or accessing the 'advanced control' parts of the home interface. Instead, the user is able to apply rules to any object (person, location, device, *etc.*) which they may currently be viewing/controlling.

Users think about problems in different ways, so obviously we need to provide users with differing means of programming logically equivalent rules for the home. (For example: "turn on the coffee machine when I get up" and "when I get up turn on the coffee machine"). This also applies to viewing rules. For this reason I have tightly integrated the rules with the home management user interface. This interface deals with people, time, locations, devices and rules. Each aspect can be viewed, controlled, and programmed through the same interface. By doing this the user is able to easily add rules to whatever component of their house they want to, and in whatever way they think about the problem.

iPad for Homer is still in development; currently the application can only support the writing and saving of policices.

5.2 Policies

Users think about the home in different ways, so they must be allowed to define logically equivalent policies in ways that suit them. For example, a device-oriented policy might say 'turn on the coffee machine when I get up, while a situation-oriented view might say 'when I get up turn on the coffee machine'. Alternative perspectives like these can also be applied to viewing policies. Policies are tightly integrated with the home management user interface. The interface reflects perspectives such as people, time, locations, devices and rules. Each perspective can be viewed, controlled, and programmed through the same interface. Users can therefore easily view existing policies. They can also define new policies for whatever aspects of the home they wish, and in whatever way they think about these.

Since users will vary in their technical abilities, they can choose different capability levels. These expose or hide various aspects of the underlying policy language. The three levels are as follows:

- Simple ('I'm a little scared'): This offers basic capabilities such that triggers and conditions can be combined only with **and**, and actions are simple lists. An example would be: **when** trigger1 **and** trigger2 **do** action1 **and** action2.
- Intermediate ('I'd like to give it a go'): This adds the capability to use **or** with triggers and conditions. An example would be: **when** condition1 **or** trigger1 **do** action1.
- Advanced ('let me do everything'): This adds the capability to combine triggers and conditions with **then**, include an associated time interval and use conditions in actions. An example would be: **when** trigger1 **then** (condition1 **or** trigger2) (occur within 2 minutes) **do** action1 **and** (**if** condition1 **then do** action 2).

Figure 10 gives an example of editing a policy at the medium level. The user gives the policy a name for future reference, specifying when the policy is triggered and what to do. Each element can be edited, reordered or removed within the current section, and can also be moved in and out of subsections for grouping. Choosing new elements can be done in multiple ways (from different perspectives) to support different user views. For example, an element that describes when the front door opens can be found through Locations (Home > Hall > Front Door > opens), Devices (Doors > Front Door > opens) or People (Someone > opens front door). These distinctions are irrelevant when saving and displaying policies, so the user can easily view the same logic from different perspectives.

For advanced users, two buttons allow saving the current set of triggers and conditions (the **when** clause/scenario) and the current set of actions (the **do** clause). As an example of a scenario, in the sample policy in Figure 10 it would be possible save the last three terms of the **when** clause as 'someone left the house'. This would then simplify the current policy and enable that scenario to be used in another policy (accessed as People > Someone > left the house). Saving the action list similarly offers a form of high-level actuator fusion. This time, a group of actions associated with an entity can be given a friendly name for use in other policies.

6 Conclusion

Smart homes need to be both controlled and programmed by anyone within the home. Having reviewed thouroughly the current state of the art of end-user programming and user interfaces for home systems, I believe there is still no single solution which offers a means of both controlling and programming the home that users like. I feel that Homer for iPad can fill this gap in both research and the commercial market, and can also provide a solution which is suitable for all users within the healthy active older generations or younger people.

6.1 Summary

Home automation and telecare have been introduced. It has been argued that a common technical approach can support both of these, the main differences being in the specific components and services. The challenges to be met include achieving acceptability, usability, interoperability and automated support. The background to home systems has been discussed in the areas of standard, platforms, automation and interfaces.

The Homer system supports both home automation and telecare. The Homer philosophy is to make home control visible to users and manageable in simple ways. A Homer component embodies some underlying device technology. Homer services provide common capabilities to components. A distinction is made between system device types (that reflect particular technologies) and user device types (that reflect how the user wishes to treat devices in the home). Devices are supported in categories such as communication, environment, home automation, telecare and user interface. The Homer framework mediates between components and the rest of the system. Component events can be mapped by event logic to/from higher-level events used at policy level. Users can define policies for how they wish the home to react to different situations. At a higher level, goals define high-level objectives that are optimally realised through policies. User-friendly interfaces allow the home to be managed through interfaces such as the iPad.

Figure 10: Rule Definition Screen on iPad

6.2 Evaluation

The Homer system represents mature work on a number of aspects:

- Service platforms are not new. Homer is based on the widely accepted OSGi framework, and so can take advantage of the stability and maturity of this as an infrastructure. To that extent, Homer has a similar basis to several other approaches such as Atlas and Saphire described in [7].
- Component architectures are also not new. However, Homer is unusual in offering an architecture that is well integrated with policy-based management (which has previously seen little use in the home). In particular, Homer components support the kinds of capabilities that make them easy to use in policies.
- The Homer policy language is broadly similar to other rule-based approaches such as Drools and Ponder discussed in [7]. A key difference is that policies for the home need to be usable by ordinary users, and need to reflect the kinds of control required for home automation and telecare. Homer thus supports distinctive forms of policies that are not found in other rule-based languages.

6.3 The Future

There are possible improvements that could be made in Homer. It would be desirable to integrate more seamlessly the Homer iPad application with the principles of policy conflicts and event logic. Although both of these aspects of Homer may be too complicated for the typical user, it could still be beneficial to offer a more integrated platform for these rules and logic to be written and to copy the notion of different user levels, to allow more novice users access to such rules. It could also be argued as desirable that all the device behaviour for an installation of Homer is made visible to the user instead of some of it being hidden at a lower level.

Writing policies is extremely difficult to make simple for users. However, added various help techniques to Homer could help make writing policies within Homer even simpler. Some ideas include the notion of the user learning what is possible as they start to write policies: slowly introducing new features and teaching the user about them with an example as and when the user is showing they are comfortable with the current set of features. Template policies could be included within the application so the user could simple walk through the policy and fill in the gaps, helping the user understand the policy format and how to create rules. There could be the concept of sharing policies with other users through a public gallery of policies which could be rated, discussed and made easily accessible. These policies could be easily tested (through a simulation of the home or maybe demonstrated in by the home itself) and saved by the user if so desired. This could help inspire users to be more creative and show them what is possible. A final idea is the idea that some people are motivated by competition and achievements, so Homer could integrate various awards, achievements and levels for the user as they successfully use the application for different tasks (e.g. write your first policy, write your first policy involving a **then** statement, write your first 10 policies, save your first **when** clause and use it in more than one policy).

Finally, full user trials should be conducted with Homer to fully explore how users interact and live with such an application.

Acknowledgments

Claire Maternaghan is supported by the Scottish Informatics and Computer Science Alliance, the University of Stirling, and the MATCH project (Scottish Funding Council, grant HR04016). Claire is grateful to her supervisor Ken Turner for his support, ideas and experience that makes Homer possible.

References

[1] Alan F. Blackwell and Rob Hague. AutoHAN: An architecture for programming the home. In *Proc. Symp. on Human Centric Computing Languages and Environments*, pages 150–157. ACM Press, New York, USA, September 2001.

[2] Scott Davidoff, Min Kyung Lee, Charles Yiu, John Zimmerman, and Anind K Dey. Principles of Smart Home Control. *UbiComp 2006: Ubiquitous Computing*, pages 19–34, 2006.

[3] Anind K. Dey, Raffay Hamid, Chris Beckmann, Ian Li, and Daniel Hsu. A CAPpella: Programming by demonstration of context-aware applications. In *Proc. Conf. on Human Factors in Computing Systems*, pages 33–40. ACM Press, New York, USA, April 2004.

[4] Krzysztof Gajos, Harold Fox, and Howard Shrobe. End user empowerment in human centered pervasive computing. In Friedemann Mattern and Mahmoud Naghshineh, editors, *Proc. 1st Int. Conf. on Pervasive Computing*, number 2414 in Lecture Notes in Computer Science, pages 134–140. Springer, Berlin, Germany, August 2002.

[5] Leonid A. Gavrilov and Patrick Heuveline. Aging of population. In Paul Demeny and Geoffrey McNicoll, editors, *The Encyclopedia of Population*, pages 27–50. MacMillan, London, UK, January 2003.

[6] Mirko Knoll, Torben Weis, Andreas Ulbrich, and Alexander Brändle. Scripting your home. In *Proc. Symp. on Human Centric Computing Languages and Environments*, number 3987 in Lecture Notes in Computer Science, pages 274–288. Springer, Berlin, Germany, May 2006.

[7] Claire Maternaghan. End of Year Report: Year 1.

[8] Claire Maternaghan. End of Year Report: Year 1, 2009.

[9] M. W. Newman, A. Elliott, and T. F. Smith. Providing an integrated user experience of networked media, devices, and services through end-user composition. In *Proc. Symp. on Human Centric Computing Languages and Environments*, number 5013 in Lecture Notes in Computer Science, pages 213–227. Springer, Berlin, Germany, May 2008.

[10] Tom Rodden, Andy Crabtree, Terry Hemmings, Boriana Kolevaand Jan Humble, Karl-Petter ʀAkesson, and Pär Hansson. Configuring the ubiquitous home. In *Proc. 6th Int. Conf. on The Design of Cooperative Systems*, pages 215–230. IOS Press, Amsterdam, Netherlands, May 2004.

[11] Timothy Sohn and Anind K. Dey. iCAP: An informal tool for interactive prototyping of context-aware applications. In *Proc. Int. Conf. on Human Factors in Computing Systems*, pages 974–975. ACM Press, New York, USA, April 2003.

[12] Khai N. Truong, Elaine M. Huang, and Gregory D. Abowd. CAMP: A magnetic poetry interface for end-user programming of capture applications for the home. In Nigel Davies, Elizabeth Mynatt, and Itiro Siio, editors, *Proc. Ubiquitous Computing*, number 3205 in Lecture Notes in Computer Science, pages 143–160. Springer, Berlin, Germany, September 2004.

[13] Kenneth J Turner. Device Services for The Home. In *NOTERE-10*, pages 41–48. IEEE, May 2010.