

The Internet Alarm Clock — A Networked Appliance Case Study

Stan Moyer, Dave Marples

Telcordia Technologies, 445 South St., Morristown, NJ 07960
stanm@research.telcordia.com, dmarples@research.telcordia.com



Performance from Experience

Copyright © 2000, Telcordia Technologies

1 Introduction

The classic PC is a multi-faceted beast, capable of performing a multitude of different functions, most of which were not even dreamed of by its original designers some twenty odd years ago. The incredible versatility of the device we've all come to use in our daily lives is also its greatest Achilles heel — so much overloaded functionality exists on this single instrument that it becomes more and more complex and difficult for the poor user to comprehend. A secondary factor is that precisely because the PC is designed to be general purpose, it is rarely optimized for any particular task that it performs day in and day out.

Increasingly cheap and abundant computing power is leading to a new type of computing device — one that is much more closely focussed on a specific task than the ubiquitous PC can ever hope to be. Norman [1] likened the difference between the PC and this new breed of device to the difference between a Swiss Army Knife and a good range of kitchen tools; a Swiss Army Knife is capable of performing an amazing range of tasks, but few would choose it above a good carving knife when it comes time to divide the Thanksgiving Turkey. Our homes are full of single purpose appliances that have evolved to do one function well, and it is reasonable to expect this specificity to come to the 'traditional' computing world too.

It is reasonable to expect the computing appliance to be able to reach into market segments that have been closed to the conventional PC, which has a household penetration of about 45% in the US [2]. In comparison there can be very few homes that do not use a microprocessor to control the washing machine, the television, video recorder or perhaps the car — computer-enabled appliances already have a penetration approaching 100%.

1.1 Goals of this work

At Telcordia we have been watching this trend towards appliances and have been investigating the implications that it has on network requirements and design for the next few years. To support this investigation and to provide a framework for it, an everyday household object, the Alarm Clock, was taken and re-designed — assuming that a relatively

unlimited amount of computing power and connectivity was available to it.

This paper reports on this work and proposes enhancements to current network architectures to facilitate appliance development.

1.2 Document Structure

We first present results from the Internet Alarm Clock experiment and some of the issues that were found, as the experiments with it became more and more complex. A solution to some of the issues that were found is then described, together with a discussion about other possible solutions to some of these issues. The final section then provides a summary of the paper.

2 The New Alarm Clock

When re-thinking the alarm clock it became obvious that there were two areas where significant improvements could be made to the traditional device;

- The user could specify what time they want to perform an action, the *Target Time* (a simple Target Time example is "arrival time at work"). The alarm clock itself calculates the 'lead time' required to achieve this goal based on *rules* that the user configures.
- With network connectivity the Alarm Clock could consider external factors in determining the correct alarm time — for example, it could see how busy your route to work is.

The first of these improvements could be made without the benefit of network connectivity, but it would be difficult, within the limited User Interface of an Alarm Clock, to make the configuration of the rules intuitive. The second improvement requires network connectivity as a cornerstone of its capability. It is worth noting that the goals of this work were not to create a commercially viable product but to investigate potential issues surrounding networked appliances. Each of these improvements is considered in more detail in the following section.

2.1 Target Time and Rule Selection

Target time and rule selection information is encoded into rules that can be stored for the clock.

Since the clock is attached to the network it can check variables implied by these rules like the weather, traffic, and flight information to see if the alarm time should be adjusted to meet the target time requirement.

This is most easily seen by an example. The user instructs the alarm clock that he wants to go the airport for an 8am flight. This is the *Target Time*. The Alarm Clock uses the airport rule (which has been previously configured by the user) to find that he likes to arrive at the airport 1 hour prior to departure. It takes 30 minutes to drive from home to the airport and one hour to get ready. Based on this information the alarm clock sets an *Alarm Time* of 5:30am.



2.2 External Factors

Variables that might still affect this calculated wakeup time are the flight status, weather conditions and the traffic on the route to the airport. The alarm clock has been configured, as part of the ruleset, to check these factors 30 minutes prior to the set wakeup time. This is called the *Check Time*. It does this by contacting a *Rule Engine* within the network, which calculates any required updates to the *Alarm Time*.

3 Implementation

This section describes the details of our implementation of the Alarm Clock network appliance.

3.1 Functional Components

Figure 1 depicts the components of the Alarm Clock as configured for this experiment. A LCD display is connected via a I²C port to a TINI board [3] which runs a Java Virtual Machine (JVM). The TINI board hosts the *Clock Driver*, which runs on the JVM and controls the display of the clock together with a *Clock Controller* process, which also runs on the JVM. This provides the logic for the user interface

on the clock and also interfaces with other network components. In earlier configurations the *Clock Controller* and *Clock Driver* processes were implemented on separate processors, which is the reason that they are distinct entities. A *Rule Engine*, implemented as a set of CGI scripts on a web server, interprets the service rules and retrieves user profile data.

The Alarm Clock is assumed to reside within an arbitrary home network, which is connected to the service provider's network through a Network Address Translator (NAT)¹. The presence of a NAT restricts the functionality that is available across this interface and places interesting constraints on the implementation of the system.

In this configuration, it is easy to see the interface to the *Rule Engine* as being the service interface, which needs to preserve consistent semantics across a number of potentially different physical endpoint devices. Thus the *Rule Engine* provides a natural demarcation between the service components in the home and in the network. It would be reasonable to expect a third party service vendor to provide this interface across a number of different Alarm Clock manufacturers equipment.

3.2 Operation

The operation of the Alarm Clock, and the functionality that is provided by the different components, is considered next. Note that it is essential that the clock has a valid IP number set for rule engine or external communication to be possible. In the current implementation this is set manually, although there is no reason why this could not be automated using DHCP.

¹ NATs are typically used by customers of service providers that limit the number of IPv4 addresses available (due to the limited number of addresses available in the IPv4 address space). NATs allow additional devices to be deployed in a home (or other) network without requiring additional IP addresses from the service provider — instead they use a private IP address space.

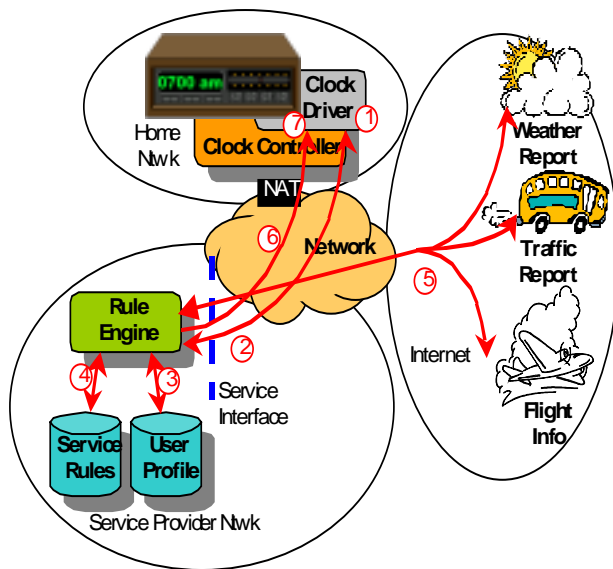


Figure 1. Internet Alarm Clock Functional Architecture

3.2.1 Setting the time

An Alarm Clock that is networked should be capable of setting the time automatically. To do this the clock makes initial contact with the *Rule Engine* when first powered up. The *Rule Engine* maintains network time synchronization using the (Network Time Protocol) NTP or a similar protocol.

3.2.2 Setting an alarm

When an alarm is to be set on the clock the *Clock Driver* establishes communication with the *Clock Controller*. It does this when the user presses any of the buttons on the front of the clock. The *Clock Controller* then takes over display I/O and communication with the user via the *Clock Driver* acting as a I/O device, in much the way that a telnet session operates.

The *Clock Controller* may, as part of its function, be required to set an alarm time. It does this by sending an Alarm Set message to the *Clock Driver*. It can also set a check time, which is the time at which the clock should establish communication with the *Clock Controller* to ensure that no external conditions have changed which would require the alarm time to be altered. Once user interaction is complete and communication between the *Clock Controller* and the *Clock Driver* is at an end the clock returns to displaying the current time, perhaps now with an *Alarm Time* and *Check Time* set.

This entire communication sequence, including the communications that occur in the rest of the network to support the transaction, is shown graphically in Figure 1 and described in the following steps.

1. The user starts a transaction by pressing one of the buttons on the clock. The *Clock Driver*, using an open protocol, opens a communication session with the *Clock Controller*. The driver and controller then work in concert to interact with the user via the buttons and display.
2. The *Clock Controller* requests the *Rule Engine*, via HTTP, to get the default *Alarm Time* and *Check Time* for the event selected for this user.
3. The *Rule Engine* retrieves the profile for the user, which contains all user's alarm rule data.
4. The *Rule Engine* retrieves service rule information and inserts the user's rule data where appropriate.
5. The *Rule Engine* executes the service rules to calculate the *Alarm Time* and *Check Time*. Execution of these rules will result in requests being made to existing web services via HTTP.
6. The *Rule Engine* returns the calculated *Alarm Time* and *Check Time* to the *Clock Controller*.
7. The *Clock Controller* sets the *Alarm Time* and *Check Time* on the *Clock Driver* and the transaction between them ends – the clock returns to displaying the current time and date.

3.2.3 Getting an Alarm Status Update

When an *Alarm Time* has been set, the clock will check, ahead of this time, that no external conditions have changed which might result in this time being adjusted. The time at which this is done is the *Check Time*. At this time the *Clock Driver* contacts the *Rule Engine* (using standard HTTP communications) to re-evaluate the current rule to ensure that no changes need to be made. This communication is instigated by the clock, thus allowing it to pass through a NAT or firewall without problems.

3.3 Results

It was found that the alarm clock offered useful functionality when implemented in this fashion. The biggest single problem was the changing format of the web sites used as information sources — a

problem that would go away if contractual relationships were entered into with information providers.

4 More Complex Scenarios

This Alarm Clock experiment shows that Networked Appliances are close to becoming a reality — they can be built with relatively inexpensive parts and can leverage existing protocols and data sources to deliver useful functionality. The experiment did raise concerns, however, that problems would emerge as more complex functionality is demanded from these networked devices. To investigate this, light and appliance control was added to the previous scenario.

4.1 Description

A Light and Appliance Controller (LAC) was added to the previous experiment. The LAC is a web server with a URL Programming Interface (UPI) [4] that serves as a front-end proxy to X.10 devices [5][6]. With the LAC any HTTP capable device can control X.10 modules via a web browser. A capability to turn on devices at times relative to the alarm time was then added to the Alarm Clock, using this LAC. Figure 2 depicts the resulting ‘composed service’ architecture.

A scenario was created where the coffee maker is to start 15 minutes before the *Alarm Time* and the heat lamp in the bathroom to switch on 10 minutes after that, 5 minutes before the *Alarm Time*. (This is a real scenario designed to help one of the authors reach work on time).

To achieve this capability, functionality was added to the *Rule Engine*² for *Actions*. The user defines the action (e.g., start coffee maker) and when it should occur. When the *Rule Engine* receives a status update it compares the current time (sent by the Alarm Clock) with the action times and, if they are equal, the *Rule Engine* initiates the action (via a request to the LAC)³.

² The fact that we were able to add additional functionality to the Internet Alarm Clock service without modifying the alarm clock device itself is an argument for putting IP Appliance functionality in network-based servers (vs. the home network or device).

³ Again, we could have had the alarm clock contact the coffee maker directly, but that requires modification of the code in the alarm clock, and that the alarm clock know how to locate, address, and contact all devices, like the coffee maker, in the home.

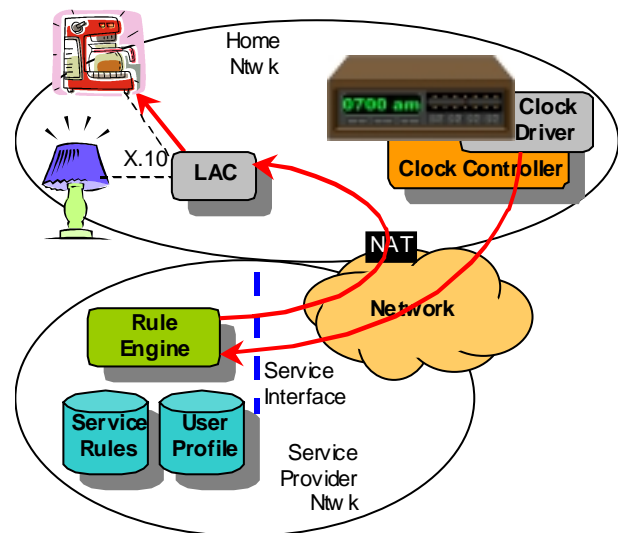


Figure 2. Service Composition Architecture

4.2 Results and Implications

In this experiment the *Rule Engine* was required to contact the LAC directly. This implies transit across the boundary of the home network, which causes several problems: external device addressability, service naming and location, and security.

4.2.1 Device Addressability

When a NAT is used to connect the home network to the outside world it is not possible to establish a connection to a device in the home from outside of the home without special configuration of the NAT, which is undesirable.

It is believed that IPv6 will start to address the available address space issue if and when it is deployed. This will cause the pressure on address space to be relieved. As a secondary benefit IPsec (which is part of IPv6) will allow communication with individual devices within the home in a secure and reliable fashion. Telcordia has developed IPv6 solutions, which allow incremental migration from a v4 to a v6 implementation with no existing service degradation during the transition, but this is not the subject of this paper.

In the short-term, IPv4 world, however, it is best to configure appliances to make outgoing calls through NATs and firewalls in order to avoid this issue (as in the first Alarm Clock experiment).

4.2.2 Service Naming and Location

There is currently no good mechanism to locate services within the home by description ('kitchen coffee maker', 'bathroom heat lamp' etc.). Without this capability, users will need an intimate knowledge of the network architecture of their home, which is obviously undesirable.

The Service Location Protocol (SLP) [7] provides a mechanism for devices to locate other devices that offer a specific service. SLP can work in a broadcast or server oriented mode, which allows a level of scalability within the home. The drawback to SLP is that it was designed to work in one domain (e.g. a single home). There is no way to specify scope and the scalability is limited.

It is noted that SLP and Lightweight Directory Access Protocol (LDAP) [8] have a similar structure. They both have parameters for a search filter, with attributes to be matched, and a request, with attributes to be returned.

4.2.3 Security

Even assuming that it is possible to communicate with the LAC within the home, it is still difficult to authenticate both ends of the communication link.

The authorization problem exists because of the desire to discriminate who may perform actions on devices within a given domain. Conventional approaches to this problem rely upon a trust relationship being established during the connection creation phase with some form of certificate or signing process then being used during the ongoing communication phase. This requires code in every endpoint to make this possible.

5 The Holistic Solution

There are solutions to each of the problems described in the previous section, and most of these solutions have already been implemented in today's networks, from NATs with port forwarding to RSA encrypted communications channels, but each of these is a piecemeal solution to the problem which lacks the elegance and cohesiveness of a complete solution.

A single architectural enhancement is proposed which addresses all of the concerns above, as well as offering several additional benefits.

The proposal is to extend the Session Initiation Protocol (SIP) [9] to accommodate the functionality required for networked appliances. SIP can already perform the 'Routing by Name' functionality that is required, rewriting the address as it progresses through the network, and it already has a reasonably complete security model.

There are, however, two small enhancements that need to be made to SIP to allow it to be used in this application. The first is that the only action that a SIP INVITE can perform is to establish a bearer between two endpoints. This is too heavyweight for short-lived connections such as those required to turn on a lamp. Instead, a new method, which we have termed KICK, is defined. This can carry payload types other than the Session Description Protocol (SDP). Typical payloads would be command sequences to be executed by particular devices within the home. KICK behaves and is routed in exactly the same way as the existing INVITE method, but it does not establish bearers as its primary function; it simply delivers its payload to the terminating SIP Proxy, which 'renders' it. The difference between KICK and INVITE is similar in many respects to the difference between Datagram and Stream messaging.

The second extension is to allow an LDAP URL [10] in the To and From fields of the SIP INVITE (and new KICK) messages, for example:
`ldap://marples.home.net/u=marples, o=Home.net, c=US, [<hostport>]"/"<dn>["?"<attributes>["?"<scope>""<filter>"]].`

The use of an LDAP URL decouples the physical address (which may or may not be IP-based) from the logical name for a device. This allows for user friendly identification of the device and hierarchical routing.

The adoption of SIP, with the enhancements above, allows for the creation of a single infrastructure for network appliance communication that:

- handles resolution of names,
- provides routing of requests,
- addresses security and authentication,
- provides for datagram messaging
- allows set-up of calls,
- and executes commands.

5.1 An Example

Figure 3 shows a user in their office, with their computer on the corporate network. The user wishes to turn off a light in their home (we only illustrate capabilities, not explain motivation). Direct communication cannot be established between the user and their home in either direction because the NAT at the access to the home and the Firewall at the access to the corporate network both preclude this.

Instead, the user sends a command to a corporate SIP Proxy, which consults its routing database to locate the SIP Proxy associated with the home network (as specified in the LDAP URL) and forwards the message to that proxy. Chances are that the name will only be partially resolved, perhaps to the user's service provider's proxy. This proxy will be able to resolve the name to the address of a SIP user agent server in the gateway device to the home and it has sufficient information available to complete the name resolution to a specific address and device code. The user agent server will then 'render' the command contained within the payload itself, transforming it to the native protocol for the destination, such as the LAC.

The SIP user agent server at the user's home will also have authenticated the SIP message originator and will have checked to ensure that the requester is authorized to execute the specified payload, thus, security is intrinsic to the approach.

The transit SIP Proxies (e.g., within the corporate network) have only provided name re-writing and ongoing routing capabilities in the context of this action and thus are relatively lightweight.

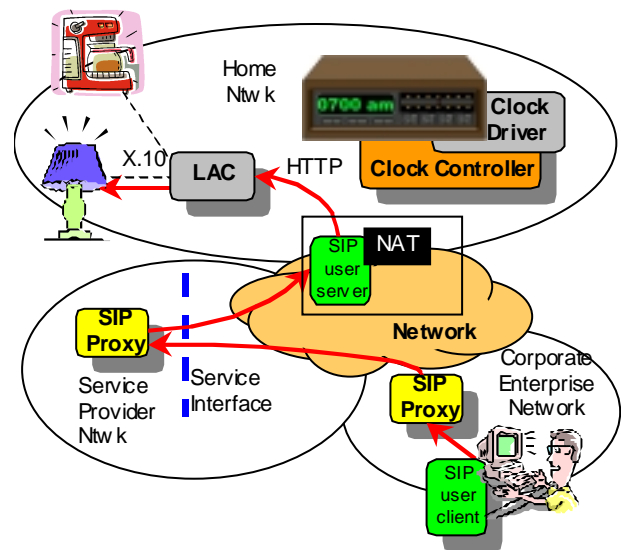


Figure 3. Example using SIP with Appliance Extensions

6 Summary

We built a simple Networked Appliance that delivers practical functionality using inexpensive off-the-shelf parts, current protocols and existing data sources, with some restrictions of connectivity and security that are intrinsic to today's network architectures.

For more complex scenarios where these restrictions become too burdensome it is possible to create an infrastructure, based around SIP with a few simple extensions, which avoids the restrictions of current networks without placing heavy demands on the capabilities of the endpoint devices.

In conclusion, it is worth noting the fact that there are many issues relating to Networked Appliances that have not yet been addressed, such as:

- Customer Care and Billing
- Provisioning and Configuration
- Service (quality) assurance/management
- Service creation/customization
- Scalability
- IPv6 migration

Telcordia is a world leader in the field of networked appliances and our research into these and other issues is ongoing.

7 References

- [1] Don Norman, *The Invisible Computer*, The MIT Press, 1998.
- [2] Reuters Business News Archives, "Two-Thirds of US Homes Online by 2003," March 29, 1999, http://www.internetnews.com/business/article/0,1087,3_13932_Ext,00.html
- [3] Introducing TINI: Tiny InterNet Interface, <http://www.ibutton.com/TINI/index.html>
- [4] Rinaldo DiGiorgio, "An Introduction to the URL Programming Interface," JavaWorld, September, 1999, <http://www.javaworld.com/javaworld/jw-09-1999/jw-09-javadev.html>
- [5] *X10 FAQ*, <ftp://ftp.scruz.net/users/cichlid/public/x10faq>, May 16, 1996.
- [6] <http://www.x10.com/support/technology1.htm>
- [7] J. Veizades, E. Guttman, C. Perkins, S. Kaplan, *Service Location Protocol, Version 2*, IETF RFC 2608.
- [8] M. Wahl, T. Howes, S. Kille, *Lightweight Directory Access Protocol (v3)*, IETF RFC 2251.
- [9] M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg, *SIP*, IETF RFC 2543.
- [10] T. Howes, M. Smith, *The LDAP URL Format*, IETF RFC 2255.