Internet Access to a Home Area Network

The AutoHan project implements a self-configuring software architecture for home area networks that offers an XML-based registry and HTTP-based event service.

Umar Saif, Daniel Gordon, and David J. Greaves
University of Cambridge, Computer Laboratory

A home area network could control many devices, all of them working together to keep your home comfortable, entertaining, and safe. But if you go out of town, you cannot watch the closed-circuit security camera you installed so that your bicycle might not get stolen—again. Secure Internet access to a home area network would let you use a Web browser or even a Web-enabled phone to control the central heating equipment, set a VCR to record TV programs, turn off an appliance that was accidentally left on, or even view snapshots from a surveillance camera.

Support for such access has at least three requirements:

- A system within the house must enable devices to export their resources to the home network and to discover and interact with other resources on it. We have implemented a system architecture called AutoHan to solve configuration and interaction problems between multiple home devices and home applications.
- These export and interaction functions need to be automated. We believe that devices in the home will eventually converge on a control architecture and that newer devices will be made with mechanisms that allow them to automatically interoperate.
- These mechanisms should lend themselves to Internet protocols and content formats to allow Internet access. This article describes one such access method.

We begin by introducing the low-level architectures of the AutoHan project that enable different networking technologies to interoperate and define one logical IP network. We then describe the two core services that enable resources to export, discover, and interoperate with AutoHan by using these low-level architectures. Finally, we discuss naming and addressing issues for Internet access and show how XML and HTTP allowed us to extend our system to support Internet access through IHan (Internet home area network).

The AutoHan System

AutoHan is our reference implementation of a home area network[1] (http://www.cl.cam.ac.uk/Research/SRG/HAN/). It con-

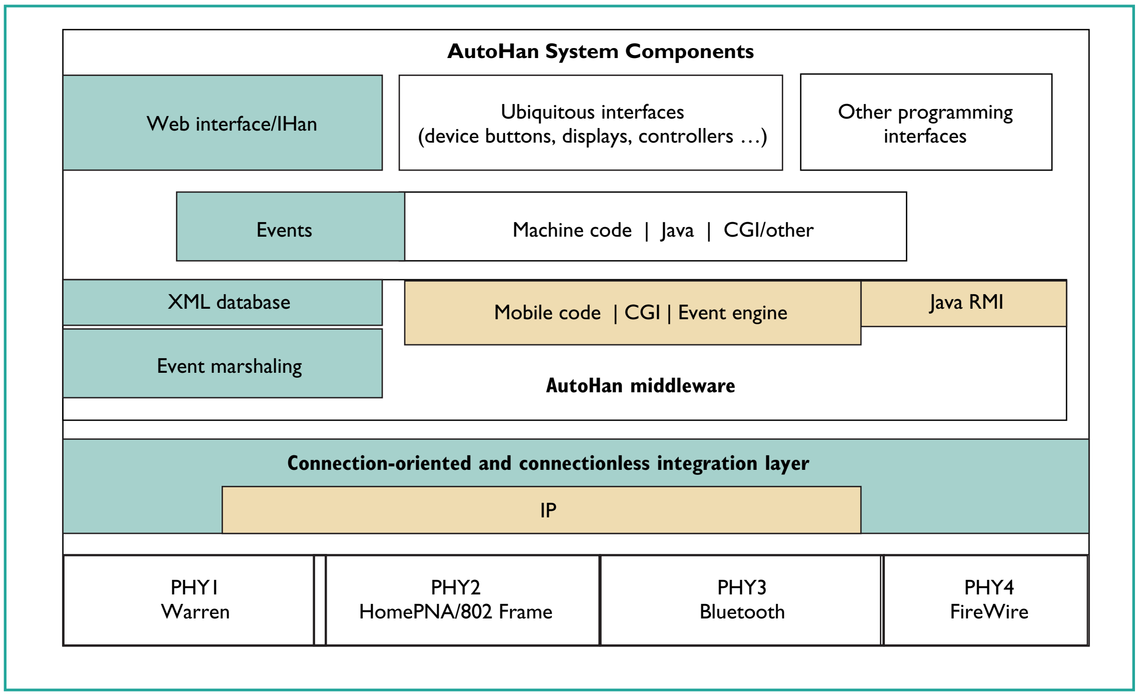
AutoHan System Components
Web interface/IHan
Ubiquitous interfaces (device buttons, displays, controllers …)
Other programming interfaces
Events
Machine code | Java | CGI/other
XML database
Event marshaling
Mobile code | CGI | Event engine
Java RMI
AutoHan middleware
Connection-oriented and connectionless integration layer
IP
PHY1 Warren
PHY2 HomePNA/802 Frame
PHY3 Bluetooth
PHY4 FireWire

Figure 1. AutoHan system architecture. Shaded components represent the design and implementation of IHan, which enables Internet access to the autoconfiguring home area network.

sists of all AutoHan-compliant devices connected to a multicast group that spans the home. A full description of the AutoHan architecture is beyond the scope of this article, but this overview describes the basic components of its operation.

As shown in Figure 1, all of the various physical layers inside the home are integrated by the bottom layer into a single network that supports both the control and data traffic of AutoHan. The AutoHan control architecture is based on events, as developed in the Cambridge Event Architecture.[2] Composite Event Engines in the network provide higher-order operations, like filtering and aggregation, on the event streams. These engines are programmed with scripts that embody the application programs that, in turn, automatically control the home on a day-to-day basis. Event scripts have the advantage of cleanly controlling the merging of applications that interact, and various pathological error cases can be automatically checked by using rules of home consistency.

In AutoHan, everything is described in XML[3]: devices, programs, rooms, people, licenses, and, of course, event types (or event schemas). Devices generate and listen to commands in the form of events corresponding to their XML schema and modify their state accordingly.

AutoHan defines three bases upon which the home network is built:

- Basic functions and control for them. The basic functions are defined in the firmware read-only memory (ROM) of a home device, together with an architecture for controlling the resources found inside that device, such as generic storage, execution, user interface, and specialized functions (for example, a DVD drive).
- Standard software devices. All home networks are expected to implement certain devices, including a registry, calendar, media storage server, voice recognizer, and an Internet gateway.
- A set of application execution environments and user interfaces. The execution environments are native machine code in device ROMs, Java programs, event scripts, and miscellaneous code via the Posix common gateway interface (CGI). The user interfaces include the buttons, displays, and infrared controllers found on home devices, and special programming interfaces in development as part of the AutoHan project (for example, Media Cubes[1]). The IHan component enables Internet access to the network (shaded components in Figure 1).

To control entities on a network requires mechanisms whereby resources can describe their control interfaces and attributes (nonfunctional), advertise these to the rest of the world, find other resources on the network, and interact with them. Two core services of AutoHan—GENA and DHan—make these actions possible.

GENA

AutoHan devices normally use the universal plug-and-play (UPnP) generic event notification architecture (GENA)[4] to send and receive events over the network via event streams based on HTTP or HTTP-UDP (user datagram protocol). GENA extends HTTP by adding three new methods: `SUBSCRIBE`, `UNSUBSCRIBE`, and `NOTIFY`. Devices can, therefore, implement a subscription arbiter, which is rather like a Web server.

Subscription arbiters, which manage subscriptions for event notifications, are used by devices that monitor or control the network, such as the AutoHan event engine. The events are generated by a wide variety of hardware devices and software entities, not all of which are compatible with AutoHan. Our prototype implementation meets this challenge by using device proxies to convert other forms of event to the GENA format. GENA defines a notification type (NT) and a notification subtype (NTS), both of which must be uniform resource identifiers (URIs). In AutoHan, the NT specifies the type of notification the subscriber requires, and the NTS gives the parameters of the event when notified by the subscription arbiter.

AutoHan devices use the UPnP generic event notification architecture to implement subscriptions.

In the standard model of GENA, an entity that wants to receive events must subscribe to the subscription arbiter and then wait for notifications. This does not fit well with the AutoHan model because, for example, an event engine may want to send as well as receive events from the same device. For simplicity's sake, the subscriber should be able to use the subscription ID by which it receives events to also generate events and pass them to the subscription arbiter. In our implementation, GENA can send events to the subscription arbiter. An alternative method could use a second subscription arbiter, per device, from which the event engine would offer subscriptions and to which the devices would have to subscribe.

Our implementation of GENA uses HTTP-UDP instead of HTTP because UDP is sufficient and appreciably more efficient for asynchronous event notifications. Lack of reliable support at the UDP layer is irrelevant, in fact desirable, for two reasons. First, most of the advanced lower layers in a home network are reasonably reliable (for example, ATM, Bluetooth, even Ethernets such as HomePNA), and time-outs and acknowledgments are a big overhead at the transport layer. Second, HTTP (and HTTP-UDP) is itself a request response protocol, making reliability easy to support with this mechanism—a classical example of end-to-end argument.[5]

DHan—The AutoHan Registry Service

DHan is our implementation's XML-based yellow-pages registry service. It allows entities to advertise their properties and control interfaces and to look up and work with other entities. Its functional model allows objects of any sort to be registered, unregistered, updated, and looked up by name or attribute list. Unlike relational directory services, DHan's XML-based information model allows storing of objects with varying numbers and types of attributes.

Each registered object is assigned a lease that, if not renewed within the leased time, automatically deregisters the object from the directory. HTTP 1.1 (and HTTP-UDP) is used as the directory access protocol. XLink and multipurpose Internet mail extension (MIME) external-body headers allow for distributed operation of the directory service, in which a lookup operation returns only a link to another entity in the directory, which might even reside with the entity itself.

An authentication scheme is used to support a security model based on an access control list.

Why XML? Entities use XML to register their resources with DHan. Network state is also represented in XML. We chose XML because it can index irregular data as a platform-independent representation of loosely matched ASCII events and because it is supported by Internet browsers. XSL[6] provides a powerful mechanism to dynamically generate sets of Web pages depending on the attributes of the registered devices and on user-defined views. XLink offers a flexible linking mechanism to allow for composite event subscriptions.

Most of the entities in a home network can be grouped hierarchically. Top-level classes include people, rooms, programs, bank accounts, event scripts, licenses, and devices. Within the device class, a potential grouping could be defined by device functionality or brand name (for example, `Display Device/Television/ColorTelevison/SonyColor Television`). This hierarchical namespace also provides a direct mapping to device modeling.[7] In this respect, XML's hierarchical namespace is a perfect match for home networks. However, most of the relationships between different entities in a home network could be sensibly expressed into alternative tag orderings within hierarchical namespace. XLink and XPointer technologies allow exactly that, and their fine-grained linking facilities provide an efficient way to describe complex relationships between different entities in a home network. For example, `Person/Owner/PayPerView/Account` could be linked to `Event/WatchMovie/PayPerView` (an otherwise unrelated hierarchy) to pay for a movie.

The competitive and evolving nature of the consumer electronics industry leads to product differentiation, newer models, and newer features. Therefore, different entities will have different numbers and types of attributes. XML is ideally suited for storing structured but irregular information of this nature, and XML's weak typing and loose matching of tag strings enables interoperation between different generations and versions of devices.

An entity in the home network, such as a closed-circuit camera, can be referenced by a fully qualified name, or *point* (somewhat similar to the distinguished names of X.500), in its hierarchical structure. A point is a concatenation of all the XML tags, starting from the root of the directory, that both identifies the object according to its place in the object hierarchy and lists attribute-value pairs that delineate the device according to its capabilities. Attributes themselves are XML tags (text nodes) and, unlike distinguished names, can occur only at leaves of a point and can assume only one value. For example, an object belonging to the object hierarchy of `Camera/StillCamera/ColorCamera/` and having the attributes of a 10-second snap frequency, a living-room location, and a 5-request capacity is the point represented below:

```
<Camera>
<StillCamera>
  <ColorCamera>
   <SnapFrequency> 10 seconds
                   </SnapFrequency>
   <Location> Living Room </Location>
   <Capacity> 5 requests </Capacity>
  </ColorCamera>
 </StillCamera>
</Camera>
```

The exact position in the hierarchy, however, is of no interest to a client that wants to look up an entity by its attributes. Therefore, the lookup operation also permits nonqualified names. A nonqualified name is composed of a partial point name only, which could be an attribute-value set, or, in an extreme case, even a single node in the point name (parallel to a relative distinguished name in X.500). Thus, the object depicted above can be referenced, though not uniquely, by a partial point name of `/ColorCamera` and an attribute list of `Location/Living Room`.

If a partial point name is used, then the register function inserts the object in the XML tree in the same subtree as any other previously registered element of the same type. A fully qualified point name belonging to no previously registered hierarchy will, naturally, create a new hierarchy. The register function always returns a fully qualified point name of the object just registered, which can then be used to refer to that object uniquely in the future.

Clearly, no one description schema can be used to group and represent entities in a home network. For efficient tree-search operations, a description schema should have minimal redundancies and minimal inter-hierarchical relationships. Our architecture defines a generic framework that can be used to support any schema standard, and we believe one will emerge as the industry agrees on different description standards, such as the resource description framework (RDF) and extensible rights markup language (XRML, http://www.xrml.org).

DHan's XML-based information model can store objects with varying numbers and types of attributes.

DHan itself is a first-class resource in AutoHan, which means that it registers itself with itself, and interacts with other devices through GENA events. Hence, it offers events itself corresponding to its access interface (for example, lease expired, new device registered, device updated). Home control platforms, such as the event engines, register themselves with the subscription arbiter exported by the DHan registry. The registry notifies the event engines of major changes, such as when a new device has been switched on and registered itself, so that event scripts can be run to support these devices. This design leverages self-organization of the network.

Suitability of HTTP 1.1 (-UDP). The home network comprises many resources with different capabilities and from different manufacturers. Combined with the requirement of ubiquitous access, this makes HTTP 1.1 (and HTTP-UDP) the protocol of choice for Internet-enabled home networks. In addition to being a universally accepted protocol, HTTP (-UDP) has many

Related Work in Home Networking

Sun Microsystem's Jini architecture[1] is closely related to our work, but IHan differs from Jini in two important ways:

- IHan uses a language-independent, text-based XML directory service with an open wire protocol, HTTP, and is therefore not bound to any one application program interface (API).
- IHan is tailored for home networking, and its directory service and access protocol focus solely on the functionality for such networks. This allows deeply embedded low-end devices in the home to support the framework. The choice of already pervasive and agreed-upon standards for home networking allows the infrastructure to scale to the Internet.

Our work has much in common with the goals of UPnP industry-standard effort (http://www.upnp.org), and our use of GENA provides basic compatibility with UPnP. However, instead of standardizing device APIs, we advocate more extensive use of events, loose matching of event tags, and emphasis on asynchronous interaction between home system components. Asynchronous interaction promises to be more scalable, reliable, and efficient when multiple applications interact and in systems with a large number of components. Our implementation leverages the use of UDP and allows an efficient end-to-end design.

Our work provides a discovery and yellow-pages service that appears to be more flexible than some protocols, such as Salutation,[2] SLP,[3] and solutions using strongly typed remote procedure calls (RPCs) based on CORBA or, of course, Jini. Other discovery protocols that use XML schemas for description of control interfaces are simple service discovery protocol (SSDP)[4] and secure service discovery service (SSDS).[5] SSDP does not provide the flexibility of a complete yellow-pages lookup service as rich as DHan's. SSDS, on the other hand, though flexible and fairly expressive, does not provide a ubiquitous lightweight directory access protocol like HTTP. SSDS design also fails to recognize that the registry service itself is a first-class resource. DHan is a first-class resource itself, and the events it offers are used to support self-organizing policies for the network.

The VESA Home Networking Committee[6] has also proposed a home network architecture based on XML. In the VESA model, each device has its current state held as an "XML page" that must be read, parsed, modified, and written back to perform control operations. These operations require considerable processing at both the client and the device. In our case, the devices are required to implement only "canned" XML (perhaps from a ROM), and a GENA interface with DHan is used to represent the network's current state. This alleviates the devices of costly XML manipulations, hence accommodating low-end embedded devices. This approach, of pushing the core home network services into a general-purpose high-end node such as the residential gateway, has also been proposed by OSGi (http://www.osgi.org), but OSGi design, like Jini, is highly Java API-centric because it is based in Java Embedded Server technology.

None of the related work here has adequately discussed naming and addressing or used a fine-grained security model, as AutoHan does. To our knowledge, AutoHan with IHan is the first architecture to provide a secure framework to control a home network from across the Internet.

References

1. W.K. Edwards, *Core Jini.* Prentice Hall, Upper Saddle River, N.J., 1999.
2. B. Pascoe, "Salutation Architectures and the Newly Defined Services Discovery Protocols from Microsoft and Sun," white paper, 1999; available online at http://www.salutation.org/whitepaper/Jini-UPnP.pdf.
3. E. Guttman et al., "Service Location Protocol, Version 2," Internet Engineering Task Force, RFC 2608 (standards track), 1999; available online at http://www.ietf.org/rfc/rfc2608.txt.
4. Y.Y. Goland et al., "Simple Service Discovery Protocol/1.0: Operating without an Arbiter," work in progress, Oct. 1999; available online at http://www.upnp.org/draft_cai_ssdp_v1_03.txt.
5. S. Czerwinski et al., "An Architecture for a Secure Service Discovery Service," *Proc. MobiCom99*, 1999; available online at http://iceberg.cs.berkeley.edu/papers/Czerwin-Mobicom99/.
6. J. DiGirolamo and R. Humpleman, "The VESA Home Network Initiative," white paper update 2; available online at http://www.vesa.org/vhnwp.pdf.

merits as a protocol to access the home network directory over the Internet.

- HTTP is lightweight (compared, for example, to a full-blown LDAP server) and, therefore, can be supported in low-end embedded devices.
- HTTP works both with underlying protocols that are connection oriented or connectionless (HTTP-UDP),[8] thus satisfying a goal of AutoHan, which is that the protocol can run before and after the network layer is set up and working.
- HTTP 1.1 methods[9] fit well with the model the IHan registry service uses for queries and updates.
- HTTP is supported by all Web browsers, leveraging access to the directory access over the Internet.
- The use of HTTP leverages Web proxy models to take load off the IHan service.
- The protocol allows new HTTP methods and MIME headers to be defined, allowing new services like GENA to be supported.

The five core HTTP methods—`GET`, `PUT`, `POST`, `DELETE`, and `HEAD`—are mapped to five event types used to interact with the directory services. GET is mapped to the lookup function of DHan: It returns the object that matches the lookup criterion encoded in the uniform resource locator (URL), along with its lease in the Date header, if the lease has not expired. Because the Date header contains the time stamp until which the registration is valid, this reply can be temporarily cached, taking load off the registry service. PUT is used to register a device, with the name encoded in the URL and attributes included in the body as XML document fragments. The Location header of the response returns the fully qualified point name with which the object was registered. The Date header gives the lease date until which the registry is valid. POST can update an existing entry by changing its attributes or by renewing the lease agreement, and DELETE unregisters an already registered object. HEAD checks only whether an object exists. For a registered object, HEAD returns the fully qualifying point name in the Location header of the response, and the lease time stamp in the Date header.

The attributes of these events are encoded as XML and MIME headers. DHan uses HTTP-UDP inside the AutoHan multicast network. For Internet access, the IHan adaptation layer converts between HTTP 1.1 and HTTP-UDP.

Sensitivity of the information in a home network warrants a prudent approach.

Security model. The security model is based on a fine-grained access control list. This list, a group membership directory, is initialized by the owner of the house, who can add, modify, or delete any entry. All other entities, including people and devices, can add new entries in this database, but can modify and delete only the group membership lists that give them such privileges. By default (when no access control attributes are registered with the tag), only the owner of the house and the entity who registered the entry can modify or delete it. Therefore, even the read permissions to "everyone" have to be given explicitly. The sensitivity of the information in a home network warrants this prudent approach.

The access control list is basically another hierarchy in DHan. The group identifiers and membership lists are stored as XML tags and are managed by the DHan service, just like all other network entities. Every XML tag in the DHan directory also contains the access permissions of the group that can access it. For example, the security hierarchy for a color camera would be `<Access Control> <Everyone/><ColorCamera/></Access Control>`. If that camera could be viewed by anyone but modified only by the owner, it would be represented as `<ColorCamera Read=Everyone Modify= ColorCamera>`.

Loosely matched hierarchical XML strings used as group identifiers leverage a much more flexible security model than, for example, a flat bit set, a la Unix. The cascading rule applies to the hierarchical entries, which can, though, be explicitly overridden. By default, access-control permissions at the root of any subtree apply to all the lower nodes, but lower nodes can apply stricter access permissions. This increases the speed of directory-access operations and allows for shorter access requests.

AutoHan Operation. With GENA and DHan in place, the home network is ready to operate. Every entity that wants to send or receive control events implements a GENA arbiter, either by itself or through a proxy. The lower levels of the network allow any new entity, such as a device like a closed-circuit camera, to notify its event arbiter of the event types it offers. The device then sends a multicast packet on the network to locate the DHan registry. Because DHan has already registered itself with itself, it returns its IP address and port address, two of its registered attributes, in reply to this UDP lookup packet. The new device can then authenticate itself via that arbiter with the directory service (or create a new group) and register its attributes, along with the location of its event subscription arbiter and the event types it offers.

The device may cause the event subscription arbiter to advertise multiple classes of event with various allowable ranges for the parameters associated with an event. Now any entity can look up this device by its attributes and, if interested, can subscribe to its events using the device's arbiter, which DHan returns as one of the device's attributes. These events are handled by embedded event handlers to control the device and/or to set up data channels to and from the device. Because the DHan registry is itself an entity and offers a few events through its arbiter (for example, entity registered, lease expired), event engines could subscribe to these events to provide higher-order control functions by using event scripts. Likewise, all composite event engines also register with DHan and run subscription arbiters.

This architecture provides a self-organizing network. Resources can discover one another by using XML lookups and can monitor and control other resources by using event streams—all without human intervention. XML descriptions (instead of, for example, strongly typed RMI interfaces) allow any resource to discover and use other resources even if it does not understand all of their functions (that is, it can ignore unrecognized tags). Conversely, automatic service degradation can be supported by one device being able to use another device that supports only partial functions expected by the first device. For example, a follow-me-video application could automatically bind itself to a range of display resources, from LCD to HDTV, and then adapt to the user's immediate area. We have also found this flexibility useful for interoperation of devices from different manufacturers. The soft state of DHan, implemented with time leases, ensures that failed or unavailable resources are automatically removed from the network. Also, the events generated by DHan can implement different self-organization policies. For example, if the central heating system fails (lease expires), also switch off the boiler.

XML descriptions allow any resource to discover and use other resources even if it doesn't understand all their functions.

Addressing and Naming

To allow Internet access to the home network, Internet Protocol version 4 (IPv4) is used for addressing—it is the most widely supported addressing scheme in the world. IPv4 also has multicast capability and is ported on almost all underlying protocols that might exist in a home network (for example, FireWire, Ethernet [HomePNA/HomePnP], Bluetooth).

Port Address Translation

A well-known problem with IPv4 is its relatively limited address space; it cannot support millions of entities in worldwide home networks. With IHan, we augment the home network with port address translation (PAT) and thus make optimal use of limited IP address space by using two techniques.

PAT at the residential gateway. This technique gives every home network a unique, permanent IPv4 address. A variant of the bootstrap protocol (BOOTP) and the ranges allocated for private internets[10] are used to assign unique IPv4 addresses to all the entities in the network. To enable Internet access to the network, the residential gateway runs the IHan module to do PAT. These port-to-IP address mappings are stored in the DHan registry by the PAT entity. The registry associates the appropriate port number with clickable events (encoded as XLinks) to represent different home devices on XML pages.

PAT at the Internet service provider. In this technique, the Internet service provider (ISP) performs PAT, mapping port numbers to IP addresses as is already done by many ISPs. Used with the first scheme, this provides two levels of PAT: one at the residential gateway and one at the ISP. Theoretically, this would allow a single IP address to be shared by 2^{32} devices in 2^{16} different houses, provided the number of active connections leaving the ISP subnet did not exceed 2^{16} per IP address. The ISP could use a private routing scheme (for example, virtual private network addressing) to make a connection to the residential gateway on demand.

Naming

Two naming schemes can also be used, corresponding to the two addressing techniques. In the first case, each house would have a unique URL that is registered with the domain naming system (DNS) and is resolved to each home network's IP address.

The second case is more interesting. In this case, only the PAT at the ISP registers itself with a unique domain name. The ISP then assigns a unique object name to each residential gateway of the houses it manages. Its mapping to a (private) IP address is stored in the DHan service running at the ISP. So, when a user quotes, for example, http://www.cl.cam.ac.uk/myhouse, http://www.cl.cam.ac.uk is resolved to the unique IP address of the ISP. A "GET myhouse" HTTP request is sent to the ISP PAT service, which looks up the private IP address assigned by the ISP to "myhouse" residential gateway, sets up a dynamic port to this looked-up IP address mapping, and forwards the request to the appropriate residential gateway DHan service. The user can then control the home network through the Internet.

Object (for example, house) names could themselves be URIs,[9] providing a large, flexible namespace. For instance, your house managed by AT&T in Manhattan could be reached at http://AutoHan.att.com/NY/manhattan/yourhouse.

IHan

IHan is a software module added to the network to enable access from the Internet. The software may be run on any suitable platform, such as a residential gateway, as shown in Figure 2. The residential gateway runs the IHan protocol stack and IHan core services. It lets a user "log in" to homes

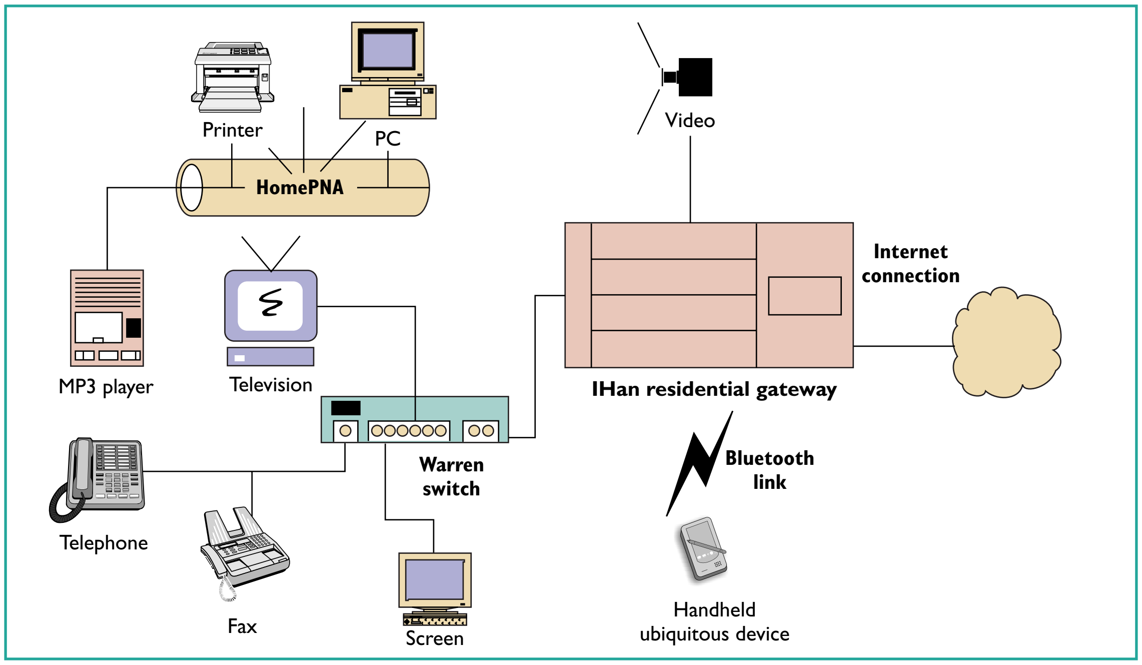

Figure 2. IHan example configuration and architecture. The IHan residential gateway runs the IHan protocol stack and core services.

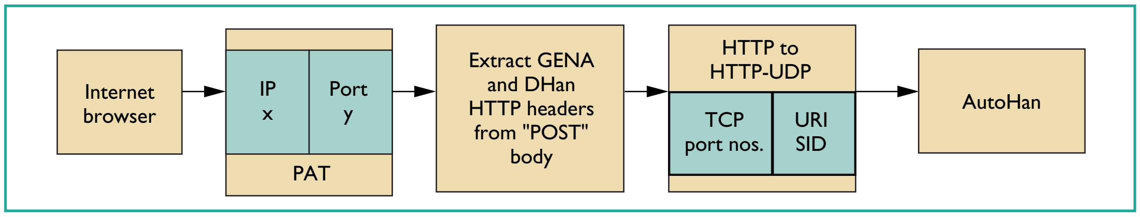

Figure 3. IHan adaptation layer. IHan exploits the HTTP "post hole" to provide compatibility with browsers.

through the Internet by quoting a URL for the IHan service and responding to an extension of RFC2617-like authentication exchange with the appropriate user name and password. IHan then presents the home network as a set of XML pages. Then, the user can find available devices and control their behavior.

IHan adaptation layer

IHan provides the necessary glue to control AutoHan from the Internet (see Figure 3). It provides the integration of connectionless and connection-oriented services by supporting both HTTP and HTTP-UDP, and exploits the "post hole" of HTTP to provide compatibility with existing browsers.

The use of XML and HTTP in the AutoHan design lends itself naturally to Web access. But current Web browsers implement only certain HTTP methods, and HTML is not detailed enough to specify when these functions need to be performed. The current Web browsers, of course, do not implement GENA headers either, and support only transmission control protocol/Internet protocol (TCP/IP).

IHan performs three major functions. First, it provides the PAT required to share an IPv4 address.

Second, it extracts the headers (GENA and others) from the "POST-encapsulated" HTTP messages. This is necessary because the HTTP entity header must be encapsulated into a URL to allow connection through a standard Web browser. The responses from a server, likewise, must be encoded in an entity body in XML. A simple encoding scheme is used—similar to that used by an HTML form to specify variable names and values to a CGI script—whereby the HTTP method, path name, and some of the MIME headers are included in the path name of a single HTTP POST request.

The GENA draft does not specify the contents of the HTTP body in the NOTIFY messages. Instead, the body contains the XML that encodes the NT and NTS header fields. IHan can use these headers to allow notification of events through a Web browser.

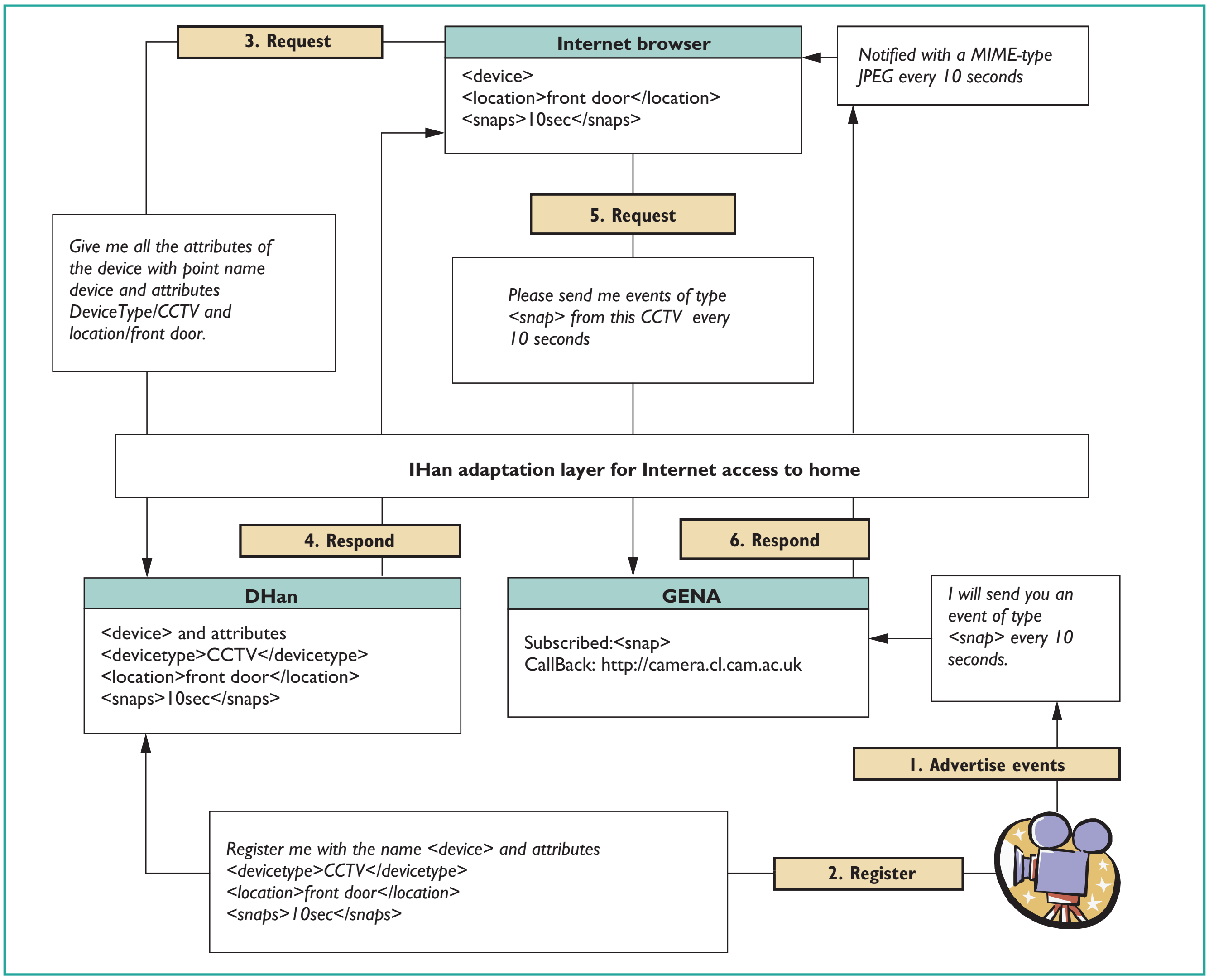
3. Request
Internet browser
<device>
<location>front door</location>
<snaps>10sec</snaps>
Notified with a MIME-type JPEG every 10 seconds
5. Request
Give me all the attributes of the device with point name device and attributes DeviceType/CCTV and location/front door.
Please send me events of type <snap> from this CCTV every 10 seconds
IHan adaptation layer for Internet access to home
4. Respond
6. Respond
DHan
<device> and attributes
<devicetype>CCTV</devicetype>
<location>front door</location>
<snaps>10sec</snaps>
GENA
Subscribed:<snap>
CallBack: http://camera.cl.cam.ac.uk
I will send you an event of type <snap> every 10 seconds.
1. Advertise events
Register me with the name <device> and attributes
<devicetype>CCTV</devicetype>
<location>front door</location>
<snaps>10sec</snaps>
2. Register

Figure 4. IHan interaction scenario. The sequence shows the exchanges that occur when a new device, a closed-circuit camera, is installed, registers itself, and is accessed from a browser.

Because HTTP is a client-pull protocol and does not fit in the event subscription and notification paradigm, this is also handled by IHan. The IHan software registers the entity and its attributes, which makes the DHan service generate XML pages with the HTML `<Auto Reload>` tag set to the frequency of the event notifications. Therefore, event notifications can now be done simply by the browser's generating an encapsulated GET request to the appropriate event arbiter every *n* sec, where *n* is the event notification frequency. Asynchronous events are currently approximated by very short reload times, but a stock-ticker-like Java applet is being coded.

Third, IHan converts HTTP TCP connections to connectionless HTTP-UDP messages, and vice versa, by maintaining a mapping between TCP port numbers of HTTP connections and HTTP-UDP unique identifiers (UIDs). When IHan receives a TCP connection, it maps the port number to a unique URI, which is then used in the UID header to relate the HTTP-UDP request/reply. HTTP serves TCP/IP Internet connections and other entities supporting connection-oriented communication, whereas HTTP-UDP receives and sends events to connectionless entities. Thus, home-networked devices need not support TCP, and commercial Web browsers are compatible with the network.

IHan Simple Example of Operation

Figure 4 shows exchanges that occur when a new device entity, a closed-circuit camera, is installed, registers itself, and is accessed from an Internet Web browser. In the example, the camera registers that it can take still snapshots, together with its resolution and frequency of snapping. It also registers its location, which in this case it has inferred by using Warren's topology-based addressing,[11] and registers the subscription arbiter that can be contacted to receive its snapshots.

The user types into the browser the URL of his home DHan service. DHan generates, and displays through HTTP, a log-in page. Once logged in, the user can query the home directory for the camera. The directory will return all entities that match the lookup criterion. The pages are rendered by using XSL, and the user can now click on any of the entity's attributes. Clicking generates an HTTP-encapsulated SUBSCRIBE request to the permanent home network IP and to the port number for the desired event arbiter. IHan removes the encapsulation headers, and the PAT software at the residential gateway forwards the request to the private IP address of the event's subscription arbiter. For instance, a user who wants "snaps events" will click on the "snap" attribute, which sends an event to the appropriate subscription arbiter, via IHan, subscribing the user's browser to picture-snap events. The page that DHan generates for this camera will reload after 10 seconds. Likewise, if the camera offers other attributes, by registering other control events with the directory service, then the user will be able to change other attributes of the camera by simply clicking on the relevant attribute and selecting the desired value (for example, change its snap frequency).

Conclusion

AutoHan defines a self-organizing home network architecture with facilities for executing and controlling programs. We have shown how the use of XML, HTTP, and GENA as the main wire protocols for entity control, together with an implementation of the IHan gateway, enables Internet access to it.

The AutoHan architecture is still in development, with emphasis on generating the event scripts for home control. From the Warren project,[11] we have a large set of networked home devices that can be incorporated into AutoHan using appropriate proxies. The current state of the system can be found on our Web page at http://www.cl.cam.ac.uk/Research/SRG/netos/han/.

Acknowledgments

We acknowledge contributions from Alan Blackwell and Andrew McNeil.

REFERENCES

1. D.J. Greaves et al., "AutoHan Services," white paper, Feb. 2000; available online at http://www.cl.cam.ac.uk/Research/SRG/HAN/AutoHAN/autohan/autohan_paper1.html.
2. J. Bates et al., "Using Events for the Scalable Federation of Heterogeneous Components," *Proc. ACM SIGOPS European Workshop*, Sept. 1998.
3. W3C recommendation, "Extensible Markup Language (XML) 1.0 (2nd ed.)," 2000; available online at http://www.w3.org/TR/REC-xml/.
4. J. Cohen, S. Aggarwal, and Y.Y. Goland, "General Event Notification Architecture Base: Client to Arbiter," work in progress, Internet draft, expired Apr. 2000; available online at http://www.upnp.org/draft-cohen-gena-client-01.txt.
5. J.H. Saltzer, D.P. Reed, and D.D. Clark, "End-to-End Arguments in System Design," *ACM Trans. Computer Science*, vol. 2, no. 4, Nov. 1984, pp. 277-288.
6. S. Adler et al., "Extensible Stylesheet Language (XSL) Version 1.0," W3C candidate recommendation, 2000; available online at http://www.w3.org/TR/xsl/.
7. Electronics Industries Alliance, "CEBus Standard EIA-600," EIA, Arlington, Va., Sept. 1996.
8. Y.Y. Goland, "Multicast and Unicast UDP HTTP Requests," Internet draft, expired Dec. 1999; available online at http://ftp.uni-bremen.de/pub/doc/internet-drafts/draft-goland-http-udp-00.txt.
9. R. Fielding et al., "Hypertext Transfer Protocol—HTTP/1.1," IETF RFC 2616, June 1999; available online at http://www.ietf.org/rfc/rfc2616.txt.
10. Y. Rekhter et al., "Address Allocation for Private Internets," IETF RFC 1918, Feb. 1996; available online at http://www.ietf.org/rfc/rfc1918.txt.
11. D.J. Greaves and R.J. Bradbury, "Warren: A Low-Cost Home Area Network," *IEEE Network*, vol. 12, no. 1, Jan. 1998, pp. 44-56.

Umar Saif is a PhD candidate at the Computer Laboratory, University of Cambridge. His research interests are in ubiquitous systems. His current research focuses on system substrates for self-organizing distributed overlays. He is working on the AutoHan project as a case study in ubiquitous device control.

Daniel Gordon received a BA in mathematics in 1990 from the University of Cambridge and a PhD in 1999 for his dissertation, "Scheduling in Optically-Based ATM Switching Fabrics." His research interests focus on low-level aspects of ATM switch design and residential ATM. On the AutoHan project, he is responsible for integration with the lower layers.

David J. Greaves is a lecturer in the University of Cambridge. He was awarded a PhD in 1989 from the University of Cambridge. In 1993, he was network architect for the Cambridge Interactive Television Trial, the world's first testbed for ATM connectivity to the home. He is a consultant to a number of companies and a founder of the Virata Corporation.

Readers may contact the authors via e-mail at {us204, dlg10, djg}@cl.cam.ac.uk.