Bionic Buffalo Tech Note #37:

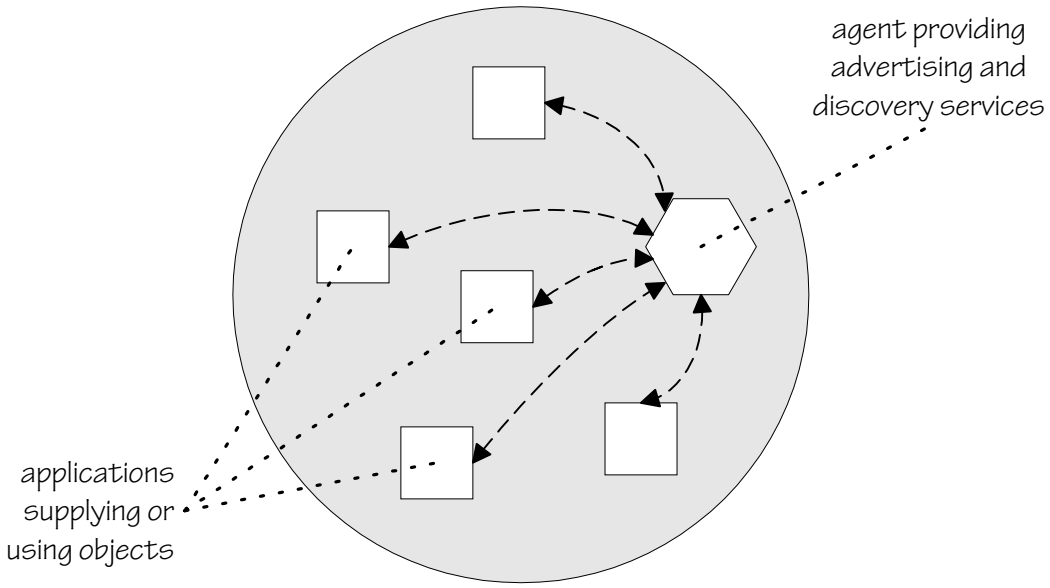
Introduction to the TOAD™ Protocol

last revised Friday 12 June 1998

INTRODUCTION

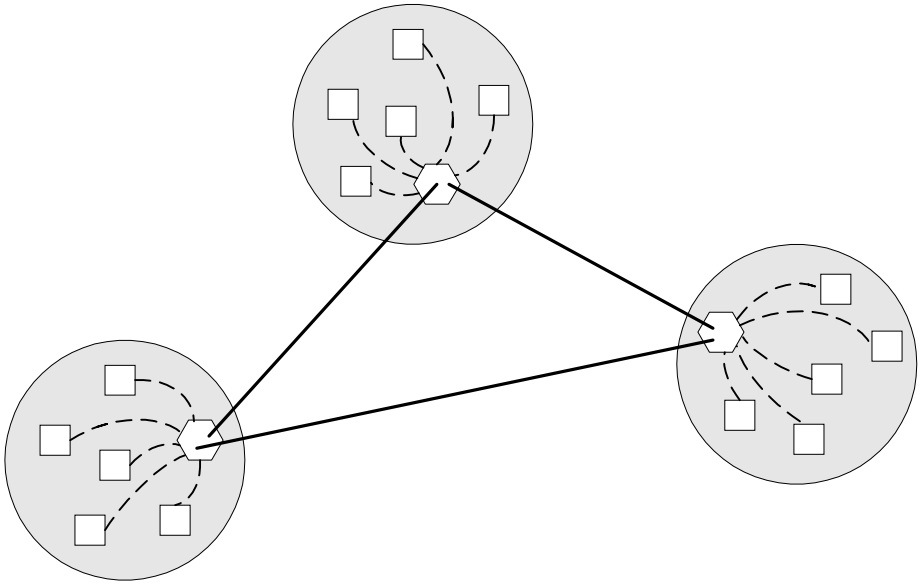
The Tractable Object Advertising and Discovery (TOAD™) protocol is used for object advertising and discovery in networked systems. Objects are entities with identities, classes, and properties. Object identities and classes are specified using arbitrary octet sequences. Object properties are typed values, using a very limited set of types.

In a standalone node, without a network, a TOAD agent offers advertising and discovery services to the applications within the node.

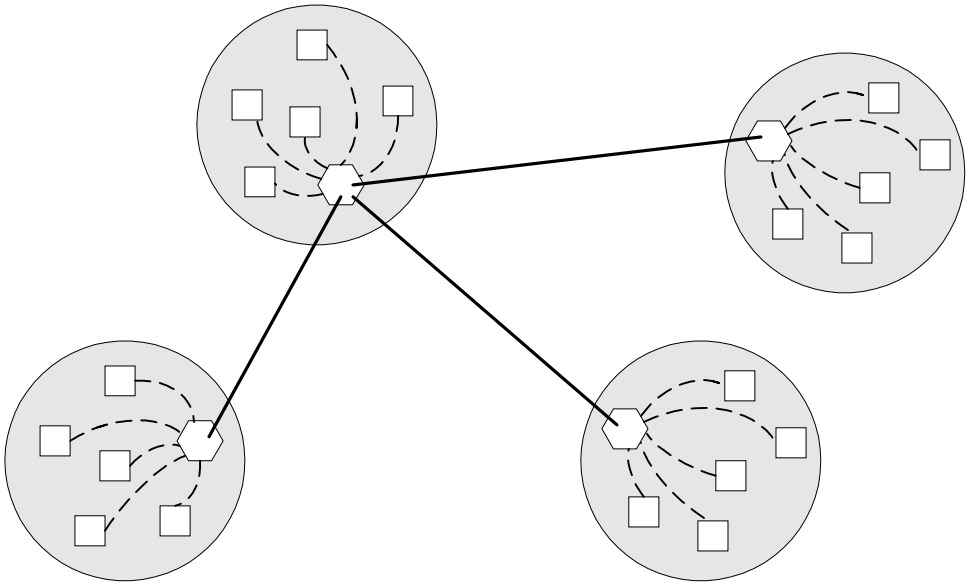


When the same node is placed on a network, the applications continue to use the same agent and the same API. The agent expands the domain of objects and classes available to applications, by combining its own node's advertised objects with those of other nodes.

On a network, agents have two functions: satisfaction of local advertising and discovery requests, and operation of the network protocols used to co-operate with other nodes. In operating systems, which expose network services to the applications, the same well-known port or access point can be used to access the agent from both inside and outside.



Although the above illustration shows a peer-to-peer relationship among the nodes, TOAD encourages dynamic topologies. The type of relationship may change during operation of the nodes. For example, a simple master-slave relationship is possible among nodes.



It is also possible to form hierarchies of two or more levels. The dynamic arrangement is controlled by management nodes, which will direct the separate TOAD nodes into relationships as needed to control the level of network traffic or meet other needs. As a node moves from a peer-to-peer role to some other role (such as slave), the network protocol it uses will change accordingly.

Furthermore, the relationship among the nodes may be different for one group of objects than for another. Two nodes might operate as peers, for instance, when serving objects of one class, while the same two nodes might operate as master and slave when serving objects of a different class.

MOTIVATION

The design goals of TOAD included:

- The protocol must be suitable for very small networks of embedded systems, but must scale *dynamically* to the Internet.
- The minimum complexity of software necessary to use the services must be very low, to accommodate simple, embedded systems without sophisticated capabilities or substantial resources.
- The protocol must be robust and resilient in the face of any unreliable node.
- The protocol must be compatible with existing protocols, and inter-operate with existing directory services.

The resulting design allows devices to operate autonomously in very small networks, but requires these same devices to defer to management in the context of larger, more complex networks.

In a single device, the implementor can choose the mechanisms for advertising and discovering objects. Object repositories might take the form of databases, or objects might be equated with file system directory entries.

In small networks, it is often practical for several such devices to use broadcast or multicast mechanisms to share repository information. Commonly, messages are sent to peers to advertise objects, or to look for objects of a given type.

As the network grows, however, the traffic and processing load from such an approach can become too great. The common approach is for the devices to use a common directory server. Besides reducing traffic, this mechanism allows groups of objects to be managed centrally in ways that are not practical among many, autonomous nodes.

Internetworking and even larger networks create additional problems, which demand still more complex protocols. These include security, performance, and management issues. The largest networks include complex tools and protocols to address problems not found in smaller environments.

The motivation behind TOAD was to allow the smallest devices to operate easily, both in small and in large networks. A TOAD-compliant device will use simple broadcast mechanisms in small networks, but will be subservient to a hierarchy when so instructed by a management node. To reduce software complexity in small devices, many message formats are common to the two behaviours: only the transmission mode (broadcast or unicast) is changed.

The changeability of the protocol from one node to another node is defined separately for the different objects in a node. A node might use one mode for certain objects, and another mode for other objects.

MODES OF OPERATION

A TOAD node operates in one of seven modes. Each mode of operation is explained in the following.

A node may operate in one mode for some of the objects it contains or seeks, and in another node for other objects.

MODE 0: QUIET MODE

In Quiet Mode, the node does not advertise or seek objects. It generates no network traffic, and fails all internal requests for discovery. (It will, however, respond to certain status requests.)

Quite Mode is entered as a result of explicit instruction, or due to some failures. The use of intentional entry into Quiet Mode is for debug purposes, or to synchronize updates of multiple objects which must have coherent versions.

MODE 1: INTERNAL MODE

When in Internal Mode, the node allows internal requests to discover and advertise objects, but keeps all information inside the node. No network traffic is generated except for responses to certain status requests.

Internal Mode effectively allows local operation, taking the node off the network.

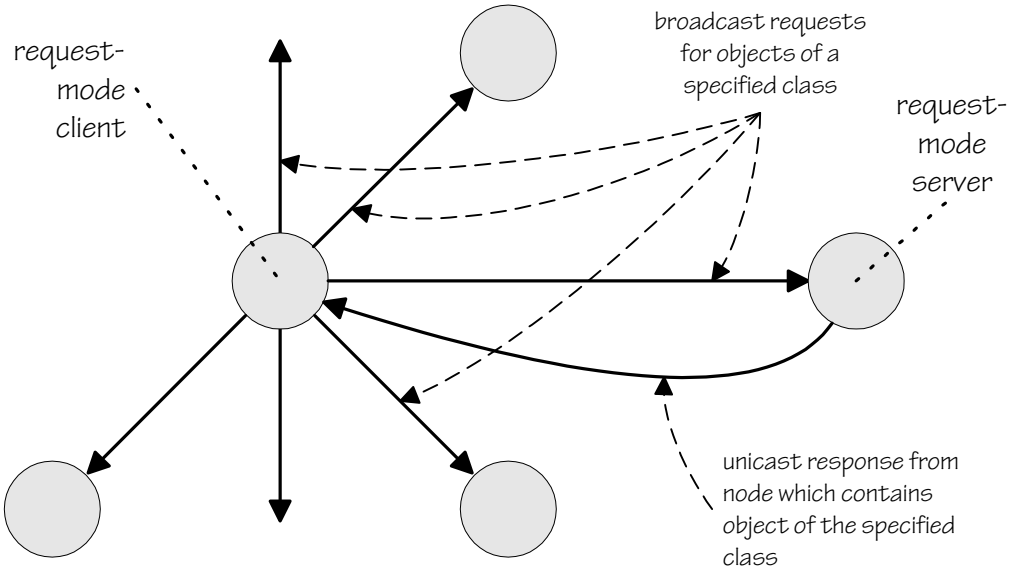
MODE 2: PASSIVE MODE MODE

In Passive Mode, the node will use information acquired by listening on the network, but will not actively seek information about specific objects. If the node learns, by listening on the network, that a certain object exists, the node may attempt to use that object. However, it will not send messages to look for an object of any particular class, or for any specific object.

When a unicast request is received from some other node for objects contained by the node, the containing node will respond to allow use of the objects. (Response is contingent upon security and other constraints.)

MODE 3: REQUEST MODE

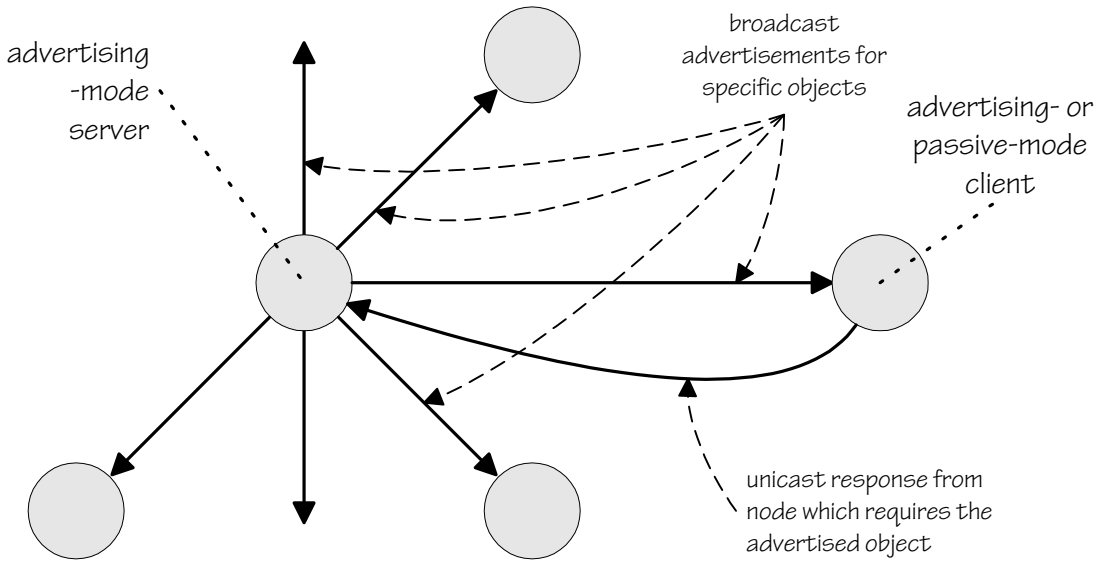
In Request Mode, the node broadcasts requests for objects of any specific class needed by the node, possibly with specific property values. It may also broadcast requests for specific objects. Broadcasts are made to a well-known port.



When a unicast request is received from some other node for objects contained by the node, the containing node will respond to allow use of the objects. (Response is contingent upon security and other constraints.)

MODE 4: ADVERTISING MODE

In Advertising Mode, the node advertises the objects it contains, by using broadcast messages sent to a well-known port. The objects are advertised by identifier and by class.



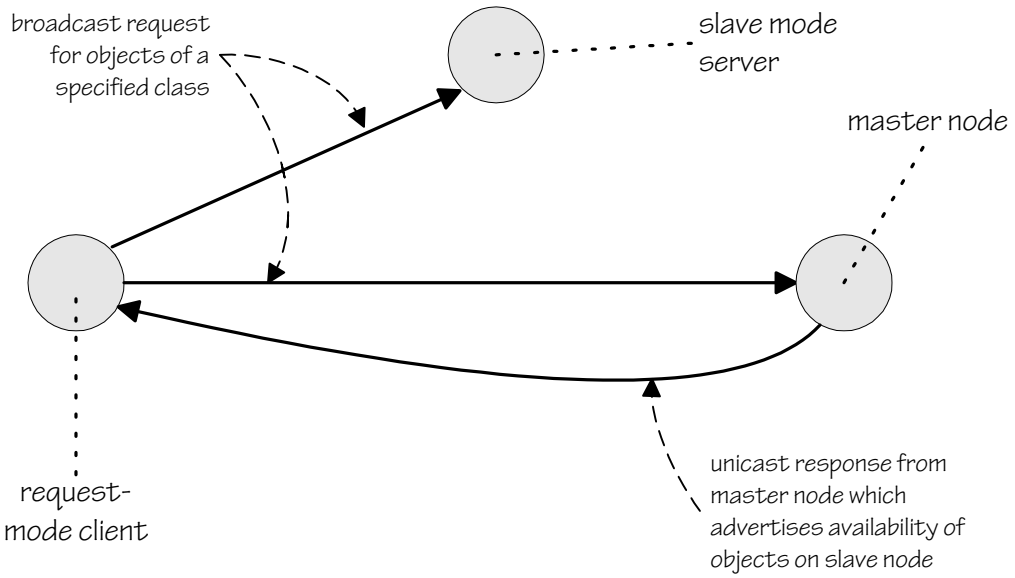
When a unicast request is received from some other node for objects contained by the node, the containing node will respond to allow use of the objects. (Response is contingent upon security and other constraints.)

The node does not advertise requirements for objects.

MODE 5: SLAVE MODE

In Slave Mode, the node subjugates itself to another node, the Master Node. The Master Node handles the affairs of the Slave Node.

In Slave Mode, the node does not respond to incoming broadcast requests. It may optionally notify its master of such requests. Incoming requests addressed specifically to the Slave are, optionally, either ignored, passed to the Master, or honoured.



To request objects of a specific class, the Slave passes the requests to its Master. The Master makes the choices for the Slave. Slave mode does not make use of broadcast transmission.

MODE 6: DIRECTED MODE

Directed Mode is similar to Slave Mode, except that the Directed Node serves multiple Masters. When a Directed Node seeks objects, it may choose from among the objects offered by its various masters.

A Directed node may be one of its own masters. In this case, the node may choose between objects contained within itself, or objects advertised to it by its other servers.

Note that there is no "Master Mode". There is no ultimate authority in this hierarchy: all authority is contextual.

Internal Mode may be used in a non-networked environment for the advertising and discovery of objects within a single node.

Request Mode and Advertising Mode are complementary. In Request Mode, the client broadcasts requests for objects of a class, whereas in Advertising Mode, the server broadcasts the availability of its objects.

Slave Mode is used for the complete subjugation of one node to another. Directed mode is similar, but gives the Directed node some measure of autonomy. If several nodes are Directed to one another as Masters, then the group operates similarly to a group of nodes in Broadcast, Aggressive, or Passive Mode, except that the network traffic uses specific addresses instead of broadcast addresses.

TRAFFIC CONTROL AND CACHING

To control the amount of traffic on the network, broadcast advertisements and requests are sent at intervals. The intervals are chosen based on the application, since some environments change more or less frequently than others. Intervals are defined by the node making the transmission, but can be altered by another node using commands in the protocol.

For example, a node may advertise at 3-second intervals by default, but this might be changed to 10-second intervals at the request of a management routine on another node.

Each node may cache information received, so the node can respond immediately to a request. Using the cache, the node does not need to wait until an advertisement or response is received. The designer of a node may optimize use of the cache, by limiting the types of information cached, the duration of caching, or other parameters.

Every advertisement, request, or other parameter is given a lifetime. The advertisement, request, or other parameter expires after the specified lifetime, or when superseded by a different value.

The limited lifetime also applies to commands to change modes. If a node is commanded to enter Passive Mode, the command will be effective only for a specified period of time. After the Passive-Mode command has expired, the node will return to the default Request Mode. Commands must therefore be renewed periodically. This prevents the network from "locking up" if a management node fails or if some portion of the network is lost.

SERVICE NEGOTIATION

The TOAD protocol serves only to advertise the existence of, or the requirement for, objects. It does not participate in negotiations for service. Once an object is located, it is up to the client to petition the service object for action.

To facilitate the negotiations among clients and servers, TOAD does permit advertising object properties. Properties are named, typed values, which are dynamically associated with specific objects. An object defines the existence and values of its own properties, using the handle returned by the object registration call.

Properties are named using simple character strings.

TOAD supports two kinds of properties: certain standard properties defined by TOAD itself, and any other properties defined by the object.

The standard TOAD properties include class, identifier, availability, usage, version, performance, and manufacturer.

Properties may be advertised with the objects, and specific property values or ranges may be associated with requests. The use of properties in advertising and requests may considerably reduce network traffic by eliminating unqualified responses.

MESSAGE FORMATS

A primary goal of the TOAD design was to reduce the level of message traffic. To this end, messages are divided into segments, each of which may deal with different objects or classes. At the beginning of the message is a simple header, which identifies the source and destination nodes.

Each message segment begins with the identifiers, classes, and properties about which that segment is concerned. These qualifiers restrict the applicability of the rest of the segment.

Optionally, an initial segment can exist without such qualifiers. In such cases, the rest of the initial segment applies to all objects and classes for the destination node or nodes.

Following the restriction (if any), is a verb identifying a function. There are four categories of function: *advertising*, *search*, *status*, and *control*.

- Advertising functions are used to announce the availability of objects. The segment prefix describes which objects are being advertised, and tells their classes and properties.

- Search functions are used to find objects. The object identifiers, properties, and classes of the objects being sought are described by the segment prefix.

- Status functions are used to ask or report on the status of the node itself, or on the objects it contains.

- Control functions are used to change the mode of operation of the destination node with respect to the objects identified by the segment prefix.

There are no functions for binding objects to clients, for negotiations among clients and servers, or for other operations expected to occur after the existence of a potential server object is known to a would-be client. Those operations are done in ways that are specific to the objects concerned.

After the function verb, the segment may contain one or more tagged fields which contain additional information about the request, the node, or the referenced objects or classes. The most important of these are:

- A *lifetime* is specified for all advertising, search, and control functions. After this lifetime, the request or availability described by the segment will expire. Indefinite operation requires periodic renewal of requests.

- A *network address* and *protocol* may be given for advertised objects, to allow would-be clients to contact the object. Depending on the protocol, the network address may be that of a broker or other agent, instead of the address of the object itself.

- *Additional properties* may describe the objects in question. Unlike the properties of the segment prefix, these additional properties are informational and do not necessarily restrict the domain of the advertisement or search. They may be used to narrow choices when several possible matches are available.

- *Protocol parameters* may be used in control or status segments to describe frequency of broadcast or other characteristics of a node's operation.

Whenever broadcast operation is used, the hop limit is set very small to preclude transmission outside of the local network and possibly one adjacent network. The use of the broadcast modes is intended for small, local networks, so there should be no need for its use on the internet. Routers should block such messages, for security reasons, since deliberate, malicious introduction of these messages into a network from the outside could pose hazards to the target.

The management of networks of TOAD nodes is allowed by the protocol, but its discussion is beyond the scope of this document.

THE USE OF OBJECTS, CLASSES, AND PROPERTIES

A main goal of TOAD design was to allow manufacturers to produce network-ready devices and appliances, which could be plugged into a LAN, home, or office network with little or no configuration by the user. To illustrate the possible applications, this section describes several examples.

For many of these applications to work to their fullest extents, the firms producing these devices or appliances must agree on the definition of object classes. This agreement will allow objects on the network to inter-operate. Each of these examples includes a discussion of the interfaces or classes, which should be defined.

Definition of common classes or interfaces may be done by any convenient means, and presupposes a larger object architecture such as CORBA. The TOAD design was done to allow easy interoperation with CORBA and CORBA Services, although other object architectures might be used.

Specification of an object class or interface in CORBA requires creating the Interface Definition Language (IDL) description of the objects. The behaviour of the objects' methods must be described using a separate mechanism.

TOAD standardizes the use of object identifiers and class names by adhering to the convention that each such identifier or class name is prefixed with an *authority* indicator. For example, CORBA uses text strings to name object classes. In TOAD, an object class name based on CORBA consists of the CORBA authority prefix, followed by the CORBA text string name of the class.

Now, the examples:

AUTOMOBILE

Although various standardized networks are used in automobiles, the objects on those networks usually have proprietary interfaces. This precludes inexpensive aftermarket addition or replacement of components in the automobile, since each component must be designed for a few specific models of automobile.

Suppose the automobile's lights, horn, engine, sound system, and console were networked, and defined using standard object interfaces.

Adding an alarm to such a vehicle would be easier than without such standardization. The alarm could discover the console (to provide a user interface, possibly as a window on a generic display), lights and horn (to flash or sound in case of violation), engine (to shut down when there is no authorization), and sound system (to provide voice interfaces or warning messages). Little or no rewiring would be required.

Adding a cellular phone would also be easier. The cellular phone could discover the radio, for example, to reduce its volume during calls. The existing sound system might be used for audio communication with the vehicle's occupant.

These and other components could be added or replaced, regardless of differing network topologies in the automobile. For instance, one vehicle design might have separate sound systems for front and rear compartments. A TOAD-compliant manager node could arbitrate requests for resources among such multiple sound systems, and the original component need not have anticipated its use in such an environment.

A simple router (possibly in an external PC) would allow connection of the vehicle to the internet, for remote diagnostics, maintenance, or software upgrades. This function is enhanced by use of the TOAD protocol, since it allows operations such as outright replacement (on the network) of an automobile's components by alternative, remote software components.

Without an object discovery protocol (such as TOAD), the automobile would have to contain a broker or directory service at a well-known address to facilitate these functions. If well-known addresses were used for all components, the broker might not be necessary, but on-the-fly replacement of components for diagnostic or maintenance purposes would not be practical.

HOME

Although most use of home networks is contained within the home, there is increasing demand to place home components on the internet. Home computers are an obvious example, but access to the internet (or another external network) allows operations such as:

- remote control of home heating and cooling for convenience or as part of a load-management process
- interrogation or control of intrusion, fire, and other alarm systems
- software upgrades and maintenance of home appliances
- remote reprogramming of appliances such as message machines, VCRs, and lighting

Placing home devices on the internet creates various concerns relating to security, routing, network management, and traffic control. Although some home networks might be very simple with low traffic loads, others may be complex and heavily loaded with multimedia and other high-volume applications.

TOAD allows simple implementations to scale to the internet. For example, consider a two-node system consisting of an intelligent thermostat and a furnace. Using simple broadcast mechanisms, the thermostat applications can find the furnace object without using a directory service or central server. When a gateway with an appropriate manager is placed on the network, both devices can be fully interoperable with remote systems.

If a load-management system is added to the network, either in the home or at a remote site such as that of a utility company, then the thermostat can be slaved to the load management software using TOAD control messages. The thermostat will co-operate with the load management system when instructed. However, if the network connection is lost, then the thermostat automatically will revert to autonomous behaviour.

OFFICE

Office environments include services for document handling, telephony, scheduling, and multimedia operations, as well as the traditional functions which were among the first to be computerized. With networked office equipment, it becomes desirable for service objects to make themselves available over the network to potential clients.

There is no shortage of alternatives in the marketplace for directory services to match would-be clients to potential servers. TOAD offers unique levels of scaleability that allow office equipment to be networked without centralized directory services. It will also allow equipment to use those services as the complexity of the office network increases. In addition, TOAD is not proprietary to any particular object or computing architecture.

A service object, using TOAD, may be implemented without any other directory service protocol. An external gateway module may then take control of the object, and incorporate it into some other directory space (or spaces). The external gateway module may reside in another device, such as a personal computer or office server.

A printer, for example, can operate with a single PC, without employing a directory server, on a two-node network. Using TOAD, the PC can discover the service objects in the printer. When the printer is connected to a larger network, it can operate as part of a complex directory service, without the storms of advertising traffic that afflict many other network service architectures. Another option is to shut down TOAD entirely in a large network, putting an alternative advertising and discovery scheme in the printer itself.

MULTITASKED APPLICATIONS

If an application consists of multiple processes, on one or more CPUs, then the usual way for one process to locate the other processes is to use the standard name service and find a process with a given name. This may not be practical for any of several reasons: the process name might not be unique, there may be multiple parallel processes of a given kind, a name service may not be available or suitable, and process properties might not be available for lookup.

TOAD can be used to advertise and discover processes in situations where the common name service mechanisms are inadequate or inappropriate. In cases where several processes implement the same function (in a parallel- or multi-processing environment), properties can be used to select the process with the lowest current load or lowest predicted response time. Properties can also be used to locate a process associated with a given client or request.

DETAILED SPECIFICATIONS AND LICENSING

Bionic Buffalo offers detailed specifications, and reference implementations, for the TOAD protocol at a nominal charge. The best future for TOAD would be widespread adoption, so there are no royalties charged.

Bionic Buffalo maintains the specification, and the name "TOAD" as a trademark, to insure that the name "TOAD" is not used with non-compliant implementations.