Network Working Group
Internet Draft
Document: draft-wall-acap-vsothers-00.txt

M. Wall
Carnegie Mellon

September 1996

The Application Configuration Access Protocol in the Context of Other Internet Protocols

Status of this Memo

1. Abstract

The Application Configuration Access Protocol (ACAP) provides a client/server-based mechanism for remote access of structured list information appropriate to common uses by internet clients. This document contrasts the approach ACAP takes to the problem of remote storage of client information to the possible use of existing protocols for this same purpose.

2. Introduction: The Key Characteristics of ACAP

The Application Configuration Access Protocol provides a client/server-based mechanism for remote access of structured list information appropriate to common uses by internet clients. It is a user- and client- based approach, one we believe has unique merit.

The question arises, however: Why another internet protocol? We at

Project Cyrus at Carnegie Mellon University have tried to explain the use for the protocol, why we came to the conclusion a new protocol was required (based on years of experience in internet client/server development), and an explanation of the functional aspects of the protocol in the white paper "The Application Configuration Access Protocol and User Mobility on the Internet" (http://andrew2.andrew.cmu.edu/cyrus/acap/white-paper.html).

The legitimate question also comes up in this context, though -- couldn't the required functionality in ACAP be achieved with another, existing protocol? This document summarizes the alternative protocol-based (and some non-protocol-based) approaches available through current technology by way of illuminating the unique approach of ACAP and why we feel its functional capacities have to be implemented as a separate protocol.

The key characteristics of ACAP are:

* ACAP is designed to accommodate disconnected use

* ACAP is designed to allows server data (and data structures) to be writable by user/clients

* ACAP is designed to handle potentially (though not necessarily) large sets of data

* ACAP is designed to allow granularity in access to data through an Access Control List mechanism

* ACAP is designed to allow per-user storage of information (accommodating problems of mobile, disconnected, and "kiosk"-model users)

* ACAP is designed to allow client definition of data fields, allowing user-side flexibility

* ACAP is designed with per-user security and authenticated operation states

* ACAP is structured to enable server-side searching.

The ACAP White Paper goes into some detail as to why these are considered required features. Here we concentrate on how other protocols stack up against these features. (Table 1 below provides a checklist of key protocol features among the various approaches that have been suggested may partially accomplish ACAP's functional goals).

The other protocols and approaches to which we are comparing ACAP in this document include: LDAP (Lightweight Directory Access Protocol) and directory services in general; DHCP (Dynamic Host Configuration Protocol); SNMP (Simple Network Management Protocol); HTTP (Hypertext Transfer Protocol); DNS (Domain Naming Service); distributed filesystems, such as NFS and AFS; and traditional database-style implementations, such as SQL.

We believe in the concept of 'the right tool for the right job'. We have no love for re-implementing the wheel, but in researching the available options in the context of Project Cyrus, we discovered this particular type of precision screwdriver, and trying to get one of these other protocols -- designed for very different forms of transactions -- is like using a heavy-duty hammer or a wrench to get this particular screw attached.

Let us look, then, at the job for which each of the above approaches was intended by way of an introduction to their flaws in trying to apply them to ACAP's job.

3. Protocol Comparison Chart

Table 1
Protocol Characteristic Chart
Internet Client/Server Data Access - Protocols and Approaches

	PROTOCOL/APPROACH							
FEATURE	ACAP	LDAP, et al	DHCP	SNMP	HTTP	DNS	NFS, AFS et al	Data-bases
Disconnected use	Yes	No	No	No	No	No*1	No*2	No*3
Client-writable	Yes	Yes	No	Yes*4	No	No	Yes	Yes
Potentially Large	Yes	Yes	No	?	No	No	Yes	Yes
Access Control List	Yes	Yes	No	Yes	No	No	Yes	No
User Storage	Yes	No	No	No	?	No	Yes	No
Client-Definable	Yes	No	No	No	?	Yes*5	Yes	Yes
Per-user Security	Yes	Yes	?	No	?	No	Yes	Yes
Server-searching	Yes	Yes	No	?	No	No	No*6	Yes
client/server	Yes	Yes	Yes	Yes	Yes	Yes	No	No
non-proprietary	Yes	Yes	Yes	Yes	Yes	Yes	No	No

Yes = has this characteristic
No = does not have this characteristic
? = not fully implemented or unclear if this could support this feature
* = qualified yes or no, see footnote

This chart addresses capabilities, not necessarily typical use
(such as SNMP's client-writing capability).

NOTES:

1. Only via the cache, which is static and non-authoritative.
2. Typically limited scalability; limited real use.
3. Transaction-locking models make this highly implementation-dependent.
4. The MIB is typically authoritative, however.
5. Usually used for limited local override, i.e. trusting a hosts file.
6. Some filesystems can search, but usually without much structure.
Most don't except through OS extensions, anyway.

4. Considerations of Applying Other Protocols to ACAP

LDAP and Directory Services (CSO, Whois++, etc.)

ACAP is designed with per-user and client-side control over entries in mind, while directory service protocols are designed (inherently) for server-side authority and administrative control.

Directory services are designed for fast lookup of relatively static, public data. Structures are defined by the server, and as such the server controls the administrative aspects of the client/server relationship. The authority for the database is entirely server-administered. In other words, if a client were to have a need for a non-pre-defined namespace or storagespace on a server, the server's administrator would have to re-define a field in the database. In many implementations of directory services, this cannot be done "live" and requires partial reconstruction of the database. LDAP in particular was designed for "Lightweight" access to the complex X.500 directory structures, in part as a response to the difficulty in getting viable implementations of X.500 directory services just for the base directory information for which it was intended (oft cited as being too complex for the immediate job at hand.)

Given the rapid pace at which client-side options change even within a single application, not to mention the diversity of multiple applications being used for similar tasks, the directory services approach is singularly ill-suited to supporting dynamic client-side data definitions. Extending this problem area to include the issue of handling different data types, the structured directory service is hampered even more in its ability to accommodate the diverse nature of user data.

Let's consider a couple of practical problems. Directory servers don't have per-user quotas for control of option storage space; ACAP does. None of the popular directory protocols supports disconnected operation, which is essential for typical client use patterns (ACAP does). Meaningful inheritance patterns and hierarchies -- such as site, system, group, user, etc. -- are non-existent in the directory services mentioned. Per-user identification mechanisms, where they exist at all, are cumbersome to deploy per-user on a large scale. ACAP by contrast allows fairly easy per-user credential control for thousands of users. In short, directory service protocols are missing many of the features which are fundamental to practical application configuration information.

We know of at least one vendor that is attempting to extend LDAP to include a structured option space for a single client. This approach

probably has merit for extremely well- and narrowly-defined client/server arrangements, but because it requires pre-defined cooperation between data structures on client and server side, and is limited to a single, specific client, is a special case solution. ACAP attempts to provide a generalized solution to the specific problems of internet client/server applications, rather than piggybacking (like a Swiss Army Knife) onto a less flexible protocol designed for a different purpose.

One side question that frequently comes up in comparing ACAP to LDAP and other directory services protocols is the application of IMSP, ACAP's predecessor, to the use of applications which use addressbooks. ACAP will also have obvious usefulness for this function. Rather than being a competing "directory" service, it's better to divide the universe of the semantically ambiguous phrase "addressbooks" into two distinct types and uses of data. LDAP and directory services provide authoritative, institution- and enterprise-wide data about users and "top-down" definitions of groups of users. It's somewhat analogous to the company directory or the Phone Book (the big paper thing next to your telephone.)

The use of ACAP for addressbooks (as we've discovered from several years of experience with IMSP, ACAP's predecessor, which was fully implemented for this purpose) has very different characteristics. ACAP/IMSP addressbooks are for the user's own view, organization, and annotation of their address information - "bottom up". To return to the analogy above, if LDAP et al are "Phone Books" then ACAP is the user's "black book" or "rolodex"- a personal repository, with access control and groups defined from the user/client's perspective. There are many reasons why a user might want to have a differing addressbook from the official one: quick reference, re-organization of data, renaming of individual user or group characteristics -- such as "Doofus" for an alias to the Boss' email address, "Softball team" for a quick grouping of people on the company softball team, or "Joe at Work and "Joe at Home" to distinguish between multiple email addresses in a way that makes sense to the user.

Our experience with IMSP is that one of its stunning (and unforseen) successes was the capability that allowed users to arbitrarily define and share addressbook information of this 'personal' nature -- lessons incorporated in the definition of the ACAP specification.

From a client-implementer's perspective, this is a key difference. Rather than having to know something about the server's view of the universe, the option space in ACAP is free and open to unforseen uses (allowing for namespace conventions). For example, if a client implementer wanted to include information on a user's favorite color -- so, perhaps, mail from that user appeared in that color or some

other level of service that might be too "silly" to impose on a formal database structure -- ACAP allows this information to be associated with any other dataset data in an ACAP dataset at the implementer's option.

LDAP and other directory services should be highly complimentary to ACAP and vice versa. The experience of users of the fully-implemented clients supporting IMSP -- which provides both IMSP and LDAP access -- strongly suggest that this is the case in real life. But for the purposes of user-storage and client-defined data, LDAP does not fill the need that ACAP does.

DHCP

DHCP was designed to address the specific problem of boot-time bootstrap information for a given, single machine. As the name implies, it's a protocol designed for "host configuration". All data is administrator- and server-specified. It is not intended or constructed with the features necessary for per-user (in contrast to per-machine) configuration on the "fly" as applications are launched sequentially or in parallel, often by multiple users on the same machine (also in sequence or parallel, depending on the Operating System).

We have done an implementation of a DHCP server locally and found the protocol to be wholly unsuitable for application configuration work in practice: it's not user-writable and has no working features designed to support user-writing with the full suite of features (security, access control at a granular layer, user-defined options, server vs. client override, etc.)

SNMP

SNMP was designed (originally largely within our development group at Carnegie Mellon University) for device monitoring and control - "network management". It lacks most features required for user configuration data management, and in particular the scalability of data and access models requisite for large-volume manipulation and retrieval of data. Like DHCP, it's not designed to store per-user data, and presently has little security in practice. ASN.1 MIBs produce similar problems of structural inflexibility to X.500.

HTTP

HTTP is largely structured for document access -- storage and transport with semantics ("hypertext"). Its main application -- albeit as successful an application as you could imagine -- is a specific form of document delivery, oriented to presentation of data

to users rather than interpreted use by client programs. It is presently underspecified for uses outside of HTML. While much work is being done at present to extend and solidify http, its fatal flaw in this context is a complete lack of data strucutre. Information is not intended to be machine-parseable; it's not structured, as ACAP is, for structured subsets of collections of data.

DNS

DNS is simply designed to provide a return of pairs of IP addresses and names, for the parsing and interpretation of domain and node names. For a given entity 'domain name', it can hold a set of tagged values, albeit a restricted set in practice: IP addresses, CNames, MX records. But DNS is limited to 256 tags, which have to be understood by prior agreement between client and server, so the data format is nowhere near flexible enough for ACAP-style information.

It provides an internet-wide hierarchy of unique tagged fields, with the engineering goal of providing very fast access to very small amounts of data. The typical ACAP application has no need for a net-wide hierarchy and needs moderately fast access to larger sets of data, with the data itself being much larger than a typical DNS entry. DNS typically provides a single-return value, while ACAP is intended to be almost always used for multiple returns from the server. DNS only provides "disconnected" access in the sense that data is statically cached and used in absence of contact with the server; and is also not written to be user-writable.

Distributed Filesystems (AFS, NFS, DFS, etc.)

Distributed filesystems are intended for storage of (mostly) unstructured user data. We have no small experience with building internet applications on top of distributed filesystems; our current messaging system (AMS -- see the ACAP White Paper for a further discussion) is layered on top of AFS (the Andrew File System, now owned and maintained by Transarc). Structure can be imposed onto a filesystem for the purpose of supporting an application, but of course this adds an additional layer of complexity to the client/server transaction and quite a bit of pre-configuration at all layers. Since there is no "universal" file system -- for many reasons well beyond the scope of this document -- any filesystem approach has the inherent flaw of being unsuitable for some operating systems due to the tight coupling of OS-specific filesystems with modern operating systems.

In many, many typical uses of internet client/server application use, the user has no need, interest, or access to a distributed file system. They're simply out of reach to the average user. And finally

we would note the obvious: even the more relatively "usual" distributed filesystems are proprietary, and none could be described as "standards-based". The core of our approach with ACAP has been to liberate the application, and the user, from reliance on any particular filesystem.

Distributed and Traditional Databases (SQL, Z39.50, etc.)

The popular approach of a traditional database application is not really appropriate for the purposes of application configuration. Database systems are almost entirely proprietary, even though "standard" query languages are available. Aside from some performance issues, which may simply be questions of implementation, databases do not lend themselves to disconnected operation due to their extremely high data integrity protections. Typically remote access, when done, is done through remote procedure calls, rather than protocol, with all the attendant problems of RPCs. Finally, the structure of queries in client/server databases shares many of the same characteristics of the structured directory service problem: data structure is authoritatively defined on the 'server' (database) side, requiring administrator intervention for new applications, fields, and data types.

6. Conclusions

Internet client application options are probably the most important type of configuration information to the vast majority of users of the internet. Other protocols were simply not designed to deal with this type of data and typical use, as evidenced by the lack of key features somewhat peculiar to application configuration. ACAP is a carefully-engineered IETF-style solution to the application configuration problem, rather than a retrofit of a protocol designed for another purpose.

7. Acknowledgements

Thanks to Chris Newman and Ned Freed of Innosoft, John Myers and Sam Weiler of Carnegie Mellon, and two reviewers who wished to remain anonymous for comments and suggestions, some of which have been incorporated into this document without specific attribution.

8. References

Myers, J., "ACAP", internet-drafts/draft-myers-acap-spec-00.txt

Wall, M., "The Application Configuration Access Protocol and User Mobility on the Internet", http://andrew2.andrew.cmu.edu/acap/acap-white-paper.html

9. Author's Address

Matthew Wall
Carnegie-Mellon University
5000 Forbes Ave.
Pittsburgh PA, 15213-3890

Email: wall@cmu.edu