OBJECT REPRESENTATION ON A HETEROGENEOUS NETWORK

C Gray GIRLING
Cambridge University Computer Laboratory
Corn Exchange Street, Cambridge, England

This paper documents some of the author's recent research into representation and naming in local area communication networks. It outlines a simple and practical method of providing what could be called 'network capabilities', an effort being made to keep this topic distinct from those of data security, data communication, network configurations and protocols.

1. Introduction

The provision of a general method of naming and accessing abstract objects is an essential prerequisite to the construction of a system that manipulates them. In particular a distributed operating system needs to refer to and manipulate the resources that it controls.

A capability-like approach is proposed in which the possession of a 'key' for a particular object provides access to that object, such a key being easily passed from one holder to another.

The environment (a **network**), in which this paper is set, needs to be only roughly defined: it contains a number of independent communicating entities (called **services**), each with the ability to communicate via an **address** with any other service necessary. The interface to each of these services consists of a number of **entries** each one corresponding to a different kind of request made to it.

In the above context an "independent communicating entity" is one which is able to withhold information from other services if it chooses. The mode of communication, protocols used, and the topology of such a collection of services are irrelevant to this paper. The security of data passed from one service to another is also an independent issue since the system described is implemented upon a network in which the desired level of data integrity has already been

achieved. Examples of such an environment could be as follows: a telephone system (in which a caller corresponds to a service), the postal system (in which a senders and receivers of post correspond to services), or a network of computers (in which processes (for example) correspond to services). This paper is particularly interested in the case characterized by the inability of callers on the telephone system to recognize each other's voices, postal system users to recognize each other's handwriting and processes in a network of computers to verify the origin of each other's messages.

2. Authentication and Object Representation

Authentication may be necessary for people, operating systems, and files (at least) all of which may show themselves to be "authentic" in different ways (a user because his password matches his name, a document because its name refers to a file containing it, and so on). Moreover, some of these forms of authentication (e.g. reading in a whole file to see if it corresponds to a document on hand) may be very expensive. In general it would be desirable to do this authentication only once, then to generate some sort of 'ticket' which says "this is to introduce such-and-such: which is a valid something-or-other", and then pass it around with the object, so that it would not have to be authenticated again. Such a ticket constitutes a representation for the authenticated object and is similar to a card of introduction, a bank card or even a pound note. A representation, although it is not actually the object it represents, will very often do instead (e.g. a pound note). This is very important in a computer network, because it is often impossible to move the actual object (e.g. a person or a computer) from one place to another.

It would be possible to operate a system in which representations were different for each object (e.g. a name and password for people, a document name and the contents of a file for documents), but such anarchy would be rather inconvenient for recipients; they would have to cope with a potentially large number of ways of verifying representations. A better idea would be to have some standard format for a representation; e.g. the statement "this is so-and-so", and something to prove that proposition. The statement "this is so-and-so" admits only a single authority, since either this can be proved (and so has been universally authenticated) or not - there is no question about the recipient of the representation discriminating about who says "this is so-and-so"; it must either be accepted absolutely or ignored. A more general statement would be "such-and-such says that this is so-and-so", so that there may be many authorities that authenticate objects; each recipient of a representation could decide exactly which authorities it is prepared to believe.

Two components of a representation system have been isolated:

i) Propositions of the form: "X says that this is Y"

ii) Things which prove the propositions in i)

It is easy to see how a proposition can be passed around a network; the two names, X and Y (in some standard format) would be sufficient. X here is the identity of the authority under which the proposition was made. It will be referred to as the **authentity** (from authority identity) of the representation. Something (a **key**) that is related to a proposition in such a way that its holder can see that they are associated would do as proof. This demonstrates the need for the third component of a representation system which is:

iii) A relation between the propositions in i) and the proofs in ii).

The two names, X and Y, and the key are kinds of name (Unique IDentifiers or UIDs for short) associated with an object's representation. X and Y, each object names, are permanent and so are called **PUIDs** (Permanent UIDs). The key, which is the essence of a representation, only exists when the object is being represented [1] and so is called a **TUID** (Temporary UID).

In order to be able to name uniquely all the objects that a network is likely to encounter during its lifetime, PUIDs must have a large number of bits; 64 are used in the Cambridge implementation. This does not, in any way, imply that PUIDs should be difficult to guess. Indeed, since a PUID will always refer to a single object and never be used for another, it is quite acceptable to "write them into" the code of programs etc. in the same way that one might a file name, or the name of a user.

In the same way that a pound note enables its holder to use the thing that it represents (some gold), the holder of a key is granted access to the object that it represents. Naturally, if 'the holder' of a key is to be distinguishable from all other potential holders it must possess that key uniquely. It is therefore important that a potential key holder cannot steal or guess keys belonging to a genuine key holder.

2.1 Stealing Keys

Above, all the things that communicate (the potential key holders) on a network have been grouped together under the single term "service" (there being a small distinction between "service" and "server" which is a service that serves something). If services are constrained to communicate only through the network (i.e. not through shared memory etc.), then it becomes impossible for one service to steal another's TUID (since the initiative must be taken by the giver).

---

[1] A user, for example, has a name (PUID) which always refers to him but there need only be a key representing him on a network when he is actually logged on.

2.2 Guessing Keys

A key can be made arbitrarily difficult (but not impossible) to guess simply by extracting it from a sufficiently large name space. That keys cannot be made absolutely impossible to guess is unimportant, because the chances of a key being guessed at random can be made smaller than the chances of any other part of the network failing. In particular a state can always be obtained in which the probability that a key will be guessed is smaller than the probability that the security of the network's communications will be compromised. In the Cambridge implementation TUIDs are effectively 48 bit random numbers.

Having decided on the attributes of PUIDs for propositions and TUIDs for keys the design of some relationship between authentities, PUIDs and keys is all that is necessary before a full object representation system can be produced.

3. Active Object Tables

One way to provide the relation between propositions and proofs referred to above it is to use the one that an encryption function creates between plain text and encrypted data. The relation between a key and the proposition "X says this is Y" can be set up by choosing the key to be the one for which <X,Y> will be its encrypted form. Thus a key will prove the proposition if its encrypted form is <X,Y> [2]. Note that <X,Y> cannot be "decrypted" to reveal its key.

This method suffers several disadvantages not the least of which is the expense of the encryption and decription operations. In addition, it is impossible to have more than one key for an object (a user could not log on twice and have his two instances distinguished, for example), it is impossible for a key to represent more than one object (so a user's representation could not simultaneously be valid under more than one authentity), and representations cannot be revoked (e.g. users cannot log off). A better method implements the relation in the same way that a relational data base does: as a table.

Each object active on the network will have a proposition involving its name and authentity associated with a key representing it in one of the tuples of a central table (called an Active Object Table or AOT for short). For example Fig. 1 shows an AOT in which an authority called USER has authenticated (i.e. logged on) user CGG and given him representation key K.

---

[2] Naturally, assuming private key encryption, both the encryption and decryption would be done in some central service(s) to prevent misappropriation of the encryption key.

The key, K, can be used to prove that "USER says that this is CGG" simply by checking that K, CGG and USER constitute one of the tuples (or **active object references**) in the AOT. When the representation is to be invalidated the active object reference is simply removed from the table, whereupon such a check would fail (an operation similar to revoking a capability).

3.1 Creating active object references

It is clear that there must be some control over the creation of active object references in the AOT. Otherwise it would be possible for any arbitrary service to be able to gain a representation for any object simply by creating a new tuple for it in an AOT.

To solve this problem, a request to create a new active object reference must include the representation of an authority. The AOT service will check this association before allowing a new active object reference to be created, making it impossible for an authority to create any tuples in the AOT except those marked with its name. The AOT recognises the representation of an authority by the fact that its authentity is a fixed name (called 'auth').

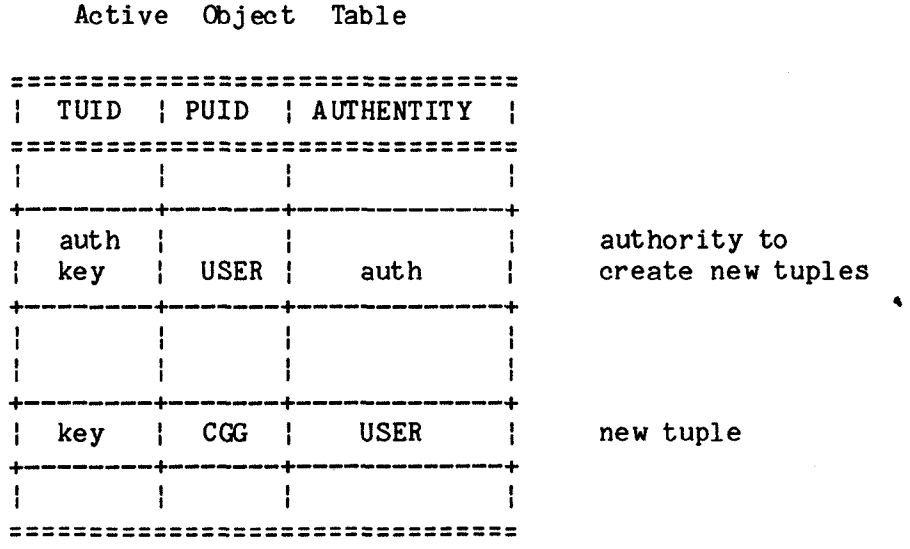

Fig. 1. A tuple for the object CGG has been created. The AOT has checked that 'auth key' and USER, which were provided with the creation request, form a tuple with the authentity 'auth', to authenticate the operation.

3.2 Deleting active object references

Deletion, in terms of AOT tuples, is trivial: the tuple corresponding to a particular PUID, TUID and authentity is removed from the table so that the key will no longer be able to prove the proposition indicated by the corresponding PUID and authentity.

As with creation, some control over the deletion of tuples from an AOT is desirable. If deletion of an entry merely required the TUID, PUID and authentity of the tuple to be quoted then any service to which the TUID of an object is passed could potentially delete it behind the original owner's back (since its PUID and authentity might be easy to guess from the context). A distinction between permission to use an object and the ownership of that object seems desirable here since, in general, the owner of an object may want only to give away permission to use a particular object rather than give it away entirely. The ability to control the existence of an object is taken here as the distinction between ownership and 'permission to use'. Accordingly an AOT tuple contains a further key, the possession of which distinguishes the owner of the object. For historical reasons this TUID-like number is called a **TPUID**. When deleting an object both its TUID and TPUID must be quoted.

TPUID	TUID	PUID	AUTHENTITY	TIMEOUT
: owner key :	: use key :	: name :	: type :	: seconds to deletion :

Fig. 2. The holder of 'owner key' can change 'seconds to deletion'. It is decremented each second and when it reaches zero the tuple is removed from the table.

In practice each AOT tuple includes a timeout value, at the end of which time the tuple will be deleted automatically. Only the possessor of the TPUID for an object can update this timeout value. Thus it is the TPUID's holder that is responsible for either maintaining the active object reference's existence or (either implicitly or explicitly) deleting it. Explicit deletion is achieved by setting a tuple's timeout to zero.

3.3 Verifying active object references

An AOT service must provide a method for checking that a particular key is a valid reference to an active object under a given authority. Indeed the AOT exists principaly to fulfil this function.

Two forms of tuple verification are provided: the first verifies that a given TUID represents a given PUID under a given authentity. The second does the same except that the tuple's TPUID is also checked. The former verifies permission to use an object and the latter verifies its ownership.

3.4 Enhancing active object references

By repeatedly creating active object references to just one object many TUIDs can be associated with a single PUID and authentity. In contrast, enhancing an active object reference enables several PUIDs to be associated with a single TUID under more than one authentity. This last type of call to an AOT service is equivalent to creating a new active object reference except that the key (TUID) for the new tuple is specified in advance. This has practical advantages when a large number of references to different objects needs to be passed about a network simultaneously since it can be accomplished using only one TUID. Thus a TUID may be present in more than one tuple and can be verifiable under a number of different names and authentities. The owner of such a multi-attributed TUID can arbitrarily control the "power" associated with it by using his TPUIDs for each of the active object references that it forms to individually control their existances.

	TPUID	TUID	PUID	AUTHENTITY	TIMEOUT
1)	tpuid1	tuid	CGG	USER	time1
2)	tpuid2	tuid	CGG	NETUSER	time2
3)	tpuid3	tuid	GRAY	USER	time3
4)	tpuid4	tuid	LABUSER	PRIVILEGE	time4
	:	:	:	:	:
5)	tpuid5	bitrep	BIT1	FACTORY1	time5
6)	tpuid6	bitrep	BIT2	FACTORY2	time6
	:	:	:	:	:

Fig. 3. Active object references can refer to different objects, even though they share the same TUID.

Enhancing an active object reference has several uses: by varying the name and the authentity associated with the TUID it can be used as a combined reference to several different objects -- e.g. 1) and 4) in Fig. 3; by varying just the name the active object can be given synonyms -- e.g. 1) and 3); and by varying only the authentity an authority (i.e. the possessor of an authentity TUID) can "confirm" an existing TUID and PUID association (to give something analogous to an unforgeable signature of approval to an active object reference) -- e.g. 1) and 2). An alternative view of enhancement is that it enables compound active objects to be created - the single TUID being the representation of a complex object consisting of several different parts, each with different names and authentities. In order to verify such an object it is necessary to verify each of its component parts -- e.g. 5) and 6).

4. AOT Service Entry Summary

The different functions explained above, constitute the entries that an AOT service must support. In summary they are as follows:

	entry	send	receive
*	gettuid	PUID,auth.t,auth.tp,AUTY,timeout	TUID,TPUID
	verify	TUID,PUID,AUTY	<nothing>
	identify	TUID,TPUID,PUID,AUTY	<nothing>
	refresh	TUID,TPUID,PUID,AUTY,timeout	<nothing>
*	enhance	TUID,PUID,auth.t,auth.tp,AUTY,timeout	TPUID

{* an entry in which a valid authentity is necessary}

In addition to whatever appears under "receive" in the above a return code is given indicating the success or otherwise of the operation. AUTY, auth.t, and auth.tp are related in the above by the fact that identify(auth.t, auth.tp, AUTY, 'auth') must succeed.

GETTUID is used to create new active object references.

VERIFY is used to prove that a key represents a particular active object.

IDENTIFY is used to prove that a key represents a particular active object representation.

REFRESH is used to maintain or delete an active object reference.

ENHANCE is used to enhance an active object reference that already exists.

5. Authority Representation

A service could obtain a TUID for any authentity it is allowed by applying to a service that has the authority to give away particular authentity TUIDs to specific services. Such a service would need to obtain a TUID for a particular authentity (call it 'auth') in order to create authentity active object references. That is, this service needs a greater authority to create lesser ones. Indeed, any particular level of authority must be delegated from a higher level. Such a process of delegation must stop at some level in the network and continue in authoritarian hierarchies outside (e.g. line management!). Within the network there need be only one place from which all authority can be delegated. This point is the Source Of All Power (**SOAP**) for the network and can be arbitrarily protected by its immediate superiors (e.g. physically, with stone walls, iron bars etc.). The details of the way in which a service might actually obtain a TUID for an authentity from the kind of server (provided by SOAP) outlined above relies upon service identification methods (which are beyond the scope of this paper).

Authentity can be used to implement the concept of 'type'. Consider a service that possesses the unique ability to create flowers and which presents an interface to the network providing flower orientated operations. This service has a valid TUID for an authentity which we shall call 'auth.flower'. Each of the active object references which the service creates in the AOT will be marked with 'flower' in its authentity field and, as long as the flower service can be trusted to create nothing but flowers (with its 'auth.flower' TUID), this authentity can be taken to indicate the (unforgeable) **type** of the objects created. New flowers which are passed about the network represented as TUIDs can always be seen to be of the type 'flower' since they will not match any AOT tuple except ones with 'flower' in their authentity field. When such a TUID is used at another flower manipulating service the fact that the object was created by 'flower' server can be verified and the inference made that this object must indeed be a flower.

PUIDs under different authentities form separate naming domains. A PUID used under one authentity need not refer to the same object that the same PUID refers to under a different authentity. For example, if a network contains a service which managed the type 'weed' weeds with a PUID 'daisy' (for example) could be generated. The flower service could equally well produce a flower of type 'flower' with the same PUID. The two objects could always be distinguished, (indeed they cannot be confused since the relevant TUIDs cannot be verified as the same object in any AOT) since one has the authentity 'weed' and the other 'flower'. Services which choose to operate only on flowers will accept only the

latter and those which choose to operate only on weeds only the former. Note that, because of the size of the PUID name space, such double use of a PUID is highly unlikely. In practice it is desirable to maintain a one to one mapping between PUIDs and objects thus avoiding ambiguous PUIDs.

Authentity TUIDs are themselves active object references distinguished by a common authentity ('auth'). The AOT service can recognize a TUID representing a valid authentity because the relevant AOT tuple will contain the PUID 'auth' in its authentity field which marks the TUID as something that allows the creation of objects in the AOT (i.e. a valid authentity).

The possessor of a TUID for the 'auth' authentity can create valid TUIDs for arbitrary authentities and hence can create and manage 'types' in the sense given above. Thus 'auth' could be considered as a the name of the type 'type'.

The logical extension from the concept of type of 'type' would be type of "type of 'type'" and so on. This path leads to a hierarchical naming scheme in which each part of a name denotes an extra layer of authority. Such a scheme can be practically produced by extending the idea of authentity to a (possibly null) list of PUIDs (i.e. a hierarchical name with one less component than the name it is forming). This type of scheme has many advantages but, in general, shares many properties of the, easier to handle, two level scheme being described. Accordingly it is not discussed further.

6. Object Representation and Secure Communication

As stated above there has been an attempt to segregate the topic of this paper (i.e. object representation) from those concerning the characteristics of network communications. This paper is concerned with what should be sent where, and not with how it is to get there. Obviously the mechanics and reliability of any particular underlying communications system will greatly influence the reliance that can be put on the mechanisms given above. No attempt, however, is being made to increase a network's security, only to provide a way of utilizing it.

The mechanisms given can be likened to a bolt on a door that represents a network's data communication system. The bolt does not make the door any stronger (even if it is stronger than the door) it only keeps the door shut: without it the door, no matter how strong it is, would provide no security at all.

Object Representation can be thought of as a network facility which can be built on top of a layer in which secure communication (using encryption, for example, or just well protected hardware) has been provided.

7. Practicability

An initial system consisting of an AOT service, a user authenticator, an authentity server, SOAP and a privilege manager has been built on the Cambridge Ring. Initial connection and file transport protocols which make use of active object references have been designed and implemented. A revision of an existing machine allocation mechanism is under way which will use them to represent network resources. However, the system is not yet in a state in which heavy regular use is made of it. This is largely due to the current lack of higher level software.

The present AOT service has room for about 500 active object references and can easily be changed to cope with more. Whether this will be necessary remains to be seen. The service itself runs on a Z80 microprocessor connected only to the Ring and is designed so that the cost of verifying references is minimal in comparison to the cost of creating them. It seems that the requesting machine's software is likely to use the lion's share of the time needed to perform these functions taken in inter-process communication and protocol software (though this is not so true of the microprocessors). The AOT resides in store for fast access. Since the maximum timeout for a representation (about six months) is large in comparison with the likely mean time between failures of the service it has proved necessary to automatically backup the table in a secret file on the Ring's file server every now and again. This has proved a success and, at the time of writing, at least one active object reference has been valid for a period of one and a half years.

The interface to AOT services and the structure of PUIDs and TUIDs has been chosen so that multiple AOT services can be provided on a network — each with a portion of the total number of object references. Whether the load on the existing service will ever justify another is also unknown.

A full evaluation of the practicability of the system will not be possible until active object orientated services have been produced in sufficient numbers to replace the current insecure access methods.