Domain Based Configuration and Name Management for Distributed Systems

K. Twidle M. Sloman

Department of Computing, Imperial College, London University,
180 Queen's Gate, London SW7 2BZ
Email: kpt@uk.ac.ic.doc or mss@uk.ac.ic.doc

ABSTRACT

As distributed systems become larger and more widely used, there is a need to structure their management to limit a manager or user's view of the components of the system. This paper describes the application of the domain concept to the structuring of configuration and name management in a distributed system. It shows how access control lists can be used as a security mechanism. Inheritance rules are specified for access permissions and access rights to cater for subdomains and overlapping domain relationships. Servers are required to maintain the domain information and location information on the components in the network. A reliable name service based on an arbitrary number of servers, which permits both partitioning and replication of information, is described.

1. INTRODUCTION

At Imperial college we have been working for a number of years on the configuration management of distributed systems. We have developed language based tools and techniques which permit distributed applications to be configured from component types. This involves creating instances of component types at specified nodes in a network, binding of the interface on one component to the interface on another to permit them to interact. One of the novel features of the Conic Configuration facilities is that it permits *dynamic configuration programming*. It is possible to perform the above configuration activities on part of the application without shutting down the whole distributed application.

Large distributed processing applications will have multiple programmers developing software at different sites. It is not practical to allow all programmers to have a single, global view of all components in the system. Instead it is necessary to limit their view of the system to a manageable set of the components relevant to the sub-system they are developing. It is also necessary to provide access controls to prevent unauthorised access to components in much the same way and for the same reasons as file systems protect files. We need to structure the management of distributed systems giving different levels of configuration access to sets of components. However this structuring must be very flexible to permit sharing of components between programmers working together on aspects of the application, and to control what components can be connected to particular services available in the distributed system.

Namespace structuring is needed to provide a local context for names. Users can then choose names for components that need only be unique within their local context. It would be very difficult to enforce globally unique names in a distributed programming environment.

It is necessary to register the names and locations of components in the system. This permits managers to keep track of the current status of components and permits users to obtain location information to bind interfaces of their new components to interfaces of existing components. However the users view of components must be limited to those which they are permitted access. This includes registering the names of components and their location to keep track of them. These functions are traditionally the role of the name or directory service.

This paper describes the use of domains as a means of structuring the configuration management and partitioning the namespace of a distributed application. A very useful feature of domains is the ability to group processes together logically for management purposes. A more detailed explanation and justification of the domain concept can be found in a companion paper [Robinson 88]. Access control has been associated with domains to provide the required protection from unauthorised configuration operations.

A centralised set of configuration management facilities would be impractical from the point of view of both performance and reliability in a large distributed system and so a decentralised configuration and name management system has been developed. The level of redundancy of these facilities can be easily adjusted to the requirements of the distributed application.

In section 2 of the paper we give an overview of the Conic environment. Section 3 describes how we use access control lists with domains as a means of providing very flexible access control for configuration management. The implementation of a reliable name server which permits both partitioning of its data as well as providing redundancy is described in section 4, followed by a conclusion.

2. OVERVIEW OF THE CONIC ENVIRONMENT

A **logical node** is the unit of configuration for building distributed applications. It is the software component which is installed on a computer and on which configuration management operations can be performed. A **node type** is specified using the Conic Configuration Language. This specifies the interface to a node in terms of strongly typed entryports and exitports through which messages are received and sent. A node can consist of a set of tasks which execute concurrently within a shared address space on a host computer as a Unix process, or directly on a bare target

computer for real-time applications. The Conic configuration language supports the concept of **hierarchical composition** in that a node type can be specified as a group of nested individual tasks or groups of tasks as shown in fig. 2.1. Individual tasks (sequential processes) are defined using the Conic Programming Language which is based on Pascal with extensions for message passing and modularity. A node contains an executive which supports multi-tasking, message passing and the operations required for dynamic configuration. The executive is itself written as a group of Conic tasks.

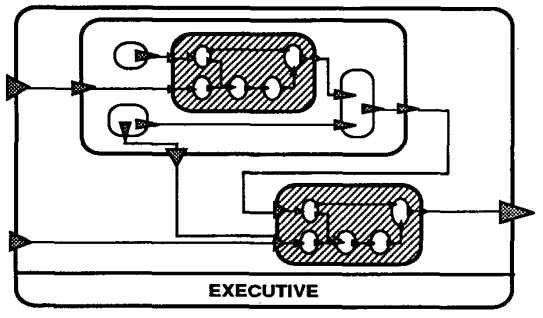


Fig. 2.1 A Logical Node

The Conic **Configuration Manager** permits a user to dynamically build a distributed application by creating instances from nodes types on the various computers available in a network [Kramer 85, Magee 87]. There can be multiple nodes running on a single host computer. Node instances are interconnected by **linking** exitports to entryports of the same type. The use of local ports provides configuration independence and permits the reuse of nodes in different configurations. The linking of ports is the means of binding interfaces.

The configuration manager is also a logical node and provides the user interface for specifying configuration changes on a distributed application. Both initial configurations and subsequent changes are specified as a declarative configuration script which is translated by the configuration manager into the set of commands to the nodes necessary to perform the operations. There would typically be one configuration manager node for each user of the system. A more detailed overview of Conic is given in [Dulay 88] and our approach to change management in [Kramer 88].

Conic is used by a number of industries, universities and research establishments as a toolkit for building distributed systems. At Imperial College it is used for teaching distributed programming to large classes of nearly 100 students, for individual student projects on distributed systems, for student group projects involving 5 students per group in a real-time laboratory as well as for funded research projects. Some programs are stand alone exercises whereas projects require integrating components produced by different programmers to form a distributed application such as the control of a model automation cell.

The hardware environment consists of multiple Sun and Vax Unix host computers which are used for software development and testing of software and a number of target M68000 computers which are used for real-time aspects of a distributed application. The computers are connected to a number of different Ethernets interconnected by both bridges and gateways. There are about 30 host and target computers which can be used for building distributed systems with plans for additional networks to be added in other departments.

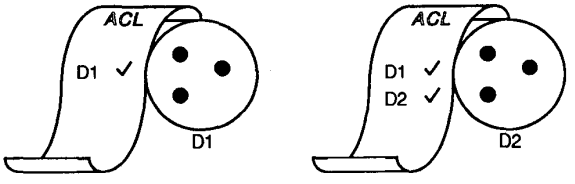
In this environment, students need access to standard services provided as part of the Conic environment, system staff need access to all parts of the system for monitoring and academic staff need access to the applications being developed by students for helping them debug etc. Conic is used for both teaching and research projects which requires the ability to structure systems for controlled sharing of selected components.

Although this is a comparatively small scale distributed environment it does exhibit many of the the requirements of large scale distributed application i.e. multiple programmers developing software on distributed computers, multiple interconnected networks, heterogeneous computers, multiple centres of administrative responsibility, as well as complex structures of shared and private components. This environment is being used to test our ideas of using domains as the basis for structuring the management of distributed systems [Sloman 87]. Our objective is to develop and test concepts which can be scaled to very large distributed systems.

3. DOMAIN BASED CONFIGURATION MANAGEMENT

3.1 Access Control Lists

A domain contains a set of nodes to which a common access control policy applies. The domain has an *access control list (ACL)* which specifies which other domains can access it and the configuration operations they can perform. An entry in an ACL gives the name of a domain from which access is permitted, and a bit mask which indicates for each configuration operation whether access is permitted or denied (see section 3.6). In the following diagrams a "√" indicates access is allowed from the named domain and an X indicates access is denied.



Domain D2 grants access to configuration managers in either D1 or D2, whereas D1 prevents any configuration from outside the domain

Fig. 3.1 Use of Access Control Lists

Note - we use the term *access permissions* of domain A to denote the information stored in a domain A's ACL i.e. which source domains (or subjects) can access domain A. *Access rights* of domain A is the set of domains it can access i.e. the object domains which have an explicit or implicit entry for domain A in their ACLs. Access permissions for a particular domain are stored in that domain's data structure, whereas access rights information is distributed in other domains.

All configuration actions are performed interactively or from a configuration script and are instigated by a Conic configuration manager which interprets user commands or the predefined script. Each user will have a configuration manager to perform configuration operations on his/her behalf. Thus there are multiple, distributed configuration managers which can perform configuration operations on a domain. This requires concurrency

control mechanisms which will not be described in this paper.

Basing the access control on domains rather than individual nodes simplifies the problem of dealing with potentially large groups of nodes which require similar access rights. For example all students in a particular year or all programmers on a particular site require configuration access to the same set of basic services. In addition, there may be similar access permissions associated with a number of service nodes. For example a set of computing resources may be treated as a unit for access control purposes - a user can access any resource in the set. The number of nodes in a domain can be limited to one or two to give fine grained access control.

3.2 Initial Domains

Each user in a system has a default *home* domain. This domain is used for creation of nodes unless otherwise specified. Users create a configuration manager node in the default domain when they log in to a workstation to perform configuration management operations. The user can create further nodes in the default domain and perform any configuration operations on these nodes and on nodes in pre-existing domains to which the user has access rights. The default domain has the same name as the user's *userid.*

A new domain is always created within another domain. There is a top level *root* domain within which default user domains are created. When a domain is created (see section 3.6.1) it has full access permissions in its ACL for the domain of the configuration node which created it. This is needed to allow the creator to modify and use the domain. A default set of access permissions can also be specified. However these access permissions can be subsequently modified.

3.3 Sub-domains

Sub-domains may be used to group a subset of nodes in a domain to which a different access control policy is to be applied or for the purposes of name visibility. Names of nodes need be unique only within a domain cf. the management of files in a tree structured file system. Users can create sub-domains within their default domains in order to provide a hierarchical structure to the nodes they wish to manage. A configuration mnager can be created in a sub-domain and a user can have multiple configuration managers.

In fact all user domains are subdomains of an enclosing domain (parent) which may correspond to the project in which they are working or to the particular year and course at Imperial College. Instead of giving access permission to individual users, we give permission to first year or third year student-domains etc. It provides a simple means of allocating access to resources to a number of people working in a team or a particular department. This requires far less entries in ACLs and requires less work if a person leaves or transfers to a different department. However it requires a means by which a sub-domain can inherit the access rights of its parent.

1. The *access permissions* granted by a domain are the union of its own ACL and the parent's *access permissions* (see fig. 3.2). The sub-domain's ACL has precedence over the parent's ACL. For example, if access is blocked in a parent but allowed in a sub-domain then access would be allowed. If access is allowed in a parent and blocked in a sub-domain then access would be blocked.

2. The *access rights* used to determine a domain's ability to access another domain are the union of its rights (based on its name) and the parent's *access rights.* In Fig. 3.3, the configuration manager in domain D3 may access domain D1 as D3 inherits the access rights of domain D2.

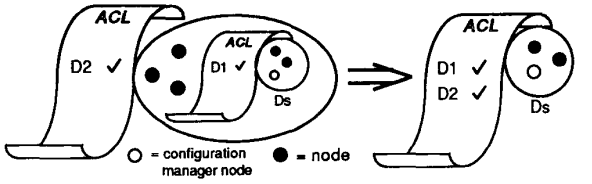


Fig. 3.2 Sub-domains Inherit Access Permissions of Enclosing Domain

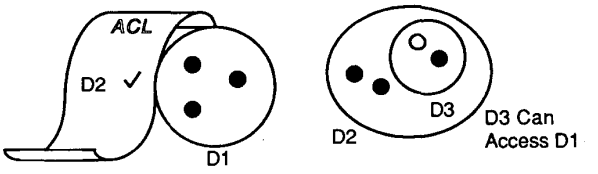


Fig. 3.3 Sub-domains Inherit Access Rights of Enclosing Domain

It is not always convenient to inherit the access rights of the parent. Fig. 3.4 represents a supervisor's domain S1 with two programmers P2 & P3 under his control. The supervisor needs to access the programmers' domains, but there is no need for P2 & P3 to access each others domains. Special entries in the ACL are used to control inheritance of access rights (IR) and access permissions (IP). In fig. 3.4, S1 blocks inheritance of its access rights (IR=X), but still allows inheritance of its access permissions (IP = √). This also prevents P2 & P3 accessing other domains to which S1 has been given access. In fig. 3.4, the blocking of D2's access to P2 takes precedence over the inherited access permission.

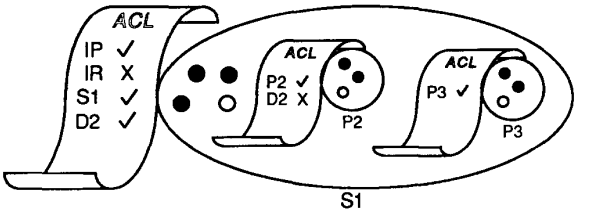


Fig. 3.4 S1 Blocks Inheritance of Access Rights

We are investigating blocking inheritance of parent's access rights at the sub-domain rather than at the parent level to give finer grained control. This permits some sub-domains to inherit the enclosing domain's rights, but others can be prevented. However we have not so far been convinced that the added complexity outweighs the benefits.

3.4 Overlap Domains

As mentioned in section 2, there is a need to permit sharing of components. Sometimes there may be a subset of the components in a domain which provide service to other domains. The users of the other domains require configuration access to the shared components. The solution is to consider the shared components to be members of an overlap domain as shown in Fig. 3.5. The overlap domain inherits the access permissions of both

D1 and D2, but does not have its own explicit access permissions.

The nodes in the overlap domain can be considered to be in both domains D1 & D2, but removing a node from the overlap domain removes it from both D1 & D2.

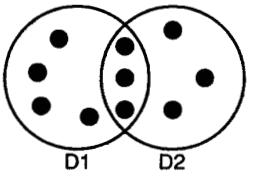


Fig. 3.5 Shared nodes in an Overlap domain

We are investigating the importing of nodes to multiple domains as an alternative method of implementing the concept of shared domains. However this results in a slightly different semantic in that removing the node from one domain still leaves it in the other domains.

3.5 References

Node and domain names must be unique within a domain i.e. a domain is a naming context. However it is possible to have a named reference in a domain to another domain. This permits the integration of disjoint domain hierarchies to permit configuration access.

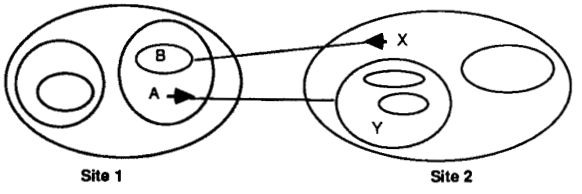


Fig. 3.6 References

Figure 3.6 shows the domains on two different sites. Domain Y on site 2 is known as A on Site 1, while Domain B on Site 1 is known as X on site 2. If the manager on site 1 removes the domain B, X is left as a "dangling" reference and access from site 2 will fail.

3.6 Configuration Operations

A user's **context** is the current set of domains which are being managed. This is analogous to multiple current directories in Unix. Operations *cannot* be specified on a domain that is not being managed. Provided the nodes's name is unique within a current context, a path name to the node does not have to be specified for an operation on that node. If there is a name clash, then the operation will fail and the user will have to qualify the name e.g. D2/N3.

3.6.1 Domain Operations

The operations which can be performed on domains include:

Manage D1 - includes domain D1 in current context

Close D1 - removes domain from current context.

Create D1 [in D2] - creates a domain D1 as a subdomain of D2. D2 must be specified if there is more than one domain in the current context. If multiple parent domains are specified, an overlap domain is created.

Remove D1 [from D2] - removes the domain D1 which must be empty i.e. not contain any nodes or subdomains.

Move N1 from D1 to D2 - moves a node from one domain to another.

Read ACL - read the access permissions stored in the ACL

Write ACL - permits changes to the ACL

Nodes - gives information on the nodes in a domain.

3.6.2 Node operations

The following configuration operations apply to a node in the current context.

Create N1:Nodetype at Sun3 - creates an instance of nodetype called N1 at computer Sun3.

Link N1. xp1 to N2.ep1 - links the exitport xp1 on N1 to the entryport ep1 on N2.

Unlink N1.xp1 from N2.ep1 - unlinks the ports.

Remove N1 - deletes the node N1

Introduce, start, stop N1 - changes the state of the node see [Kramer 88]

Status N1 - provides information on current state and interface ports of a node.

Additional operations to lock nodes for concurrency control purposes are also needed because multiple configuration managers can be performing configuration operations simultaneously. A decentralised locking mechanism, based on [Rosenkrantz 78], which prevents deadlock and starvation has been implemented.

3.7 ACL Permissions

For each of the above operations, the following access permissions may apply. Each entry in the ACL has a domain name and the access permission associated with it.

Allow: permits the operation.
Block: prevents the the operation.
Pass: this is needed for the case when some operations are to be allowed or blocked, but access permissions for others are to be inherited from the parent's ACL
Block and log: This is for use in commercial organisations where the security policy is to log every failed attempt at access.
Allow and log: for use where the security policy is to log every access to a domain.

Every ACL has a default entry which is to be applied where there is no specific entry for the subject domain.

4. NAME LOCATION

A name service is needed to maintain information domain relationships and ACLs as well as the location and status of the nodes within a domain [Magee 87]. The configuration nodes use

the service to find out what domains are accessible, and the status of nodes within those domains. Information on a node's interface is obtained directly from the node once its location is known. Nodes report status information to the server responsible for their domain.

4.1 Original System

Our initial configuration management facilities were based on a single server in a well known location. A server location file was created on all hosts containing the absolute address of the server. A server was started at that address and nodes created anywhere on the network looked in their local file for the address of the server and then sent periodic status messages to it. The single server knew about all nodes. Domains were disjoint i.e. no interaction between domains was possible and there was no access control.

This was unsatisfactory because it had a single failure point and moving the server required altering location files on all host nodes. A more flexible approach based on multiple servers which can be created anywhere in the network was needed.

4.2 Decentralised Servers

One way to overcome most of the problems would be to have all the nodes broadcast status messages to all servers. This would permit location independent multiple servers. Every server would still hold information on all nodes, thus fully replicating the data available to configuration managers. A problem with this approach is that broadcast facilities are not generally available to ordinary users on Unix. It would not be feasible for all nodes to assume superuser privileges before running, as this would constitute a gross breach of security!

An alternative considered, was to have a local server on every host. User's nodes would send their status messages to their local server (at a well known local address) and the local server could then broadcast to other servers. The problem is that local servers still rely on broadcasts and the need to have superuser privileges. Broadcasts should be avoided as they are impractical for interconnected subnetworks and cause unnecessary loading on all recipient computers which may not really be interested in the messages. Multi-destination would be a better solution, but this service is not available under Unix even though many local area networks support it.

The solution adopted was to have a set of cooperative servers at well known locations. The configuration nodes are given a set of addresses from which they should try to find a server. The servers form a logical ring and pass information about themselves and the domains they manage around the ring. The domain data base is partitioned with some replication. Two or more servers maintain information on each domain and a server can maintain information on multiple domains. A ring would normally correspond to a top-level domain at a site in a large distributed system. A higher level inter-site ring of servers would maintain information on the site domains, with one or two servers in each site domain acting as gateway servers. (See fig. 4.1). This approach is very flexible in that aditional rings can be added within a sub-domain if required. In fig. 4.1 the inter-site ring corresponds to an inter-site domain.

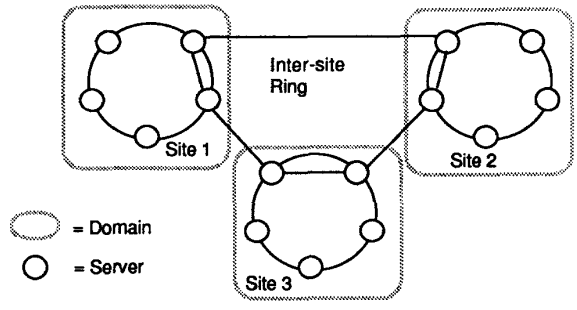


Fig. 4.1 Hierarchical Rings of Servers

4.3 Server Protocols

4.3.1 Inter-Server

Server Initialisation

We consider 4-5 servers provide adequate reliability for a site such as Imperial College which has two subnets, about 40 workstations and targets which might be used for Conic systems, and at most a few hundred nodes at any time. The servers are all in the top level (root) domain for the site. The servers in a domain form a logical ring and exchange information on their addresses, the top-level subdomains they are responsible for and whether they are acting as gateways to other top-level site domains. Servers in different domains automatically form independent rings.

When a server starts up it has to locate the other servers in its domain to form a ring. The use of broadcast was unacceptable for the reasons discussed above. Instead, a new server is given a list of addresses where another server is likely to be found. It tries each of these in turn until it finds one and is then able to join the existing ring. If no others are found, the new server assumes it is the only one in the domain and reverts to normal server role. It also periodically attempts to find other servers so as to make sure that partitioned rings are eventually rejoined. A gateway server would be given at least two separate lists of servers to try to join. Note a new server does not have to be at one of the designated locations

Information Exchanged between servers

Each server maintains an ordered list of the other active servers in the ring called the **active server list**. This contains an address for each server, the domains it is responsible for, and its load i.e. the number of nodes on which it is holding information. In addition a server maintains a **node list** which holds the location and status of each node in the domains for which it is responsible. A server which is responsible for a domain, continues to be responsible for sub-domains created within that domain, but we intend to use the load information to allocate new domains to lightly loaded servers.

The active server list is sent to the first server on the list periodically or when there is a change in the information in the list. The recipient rotates the list, updates its own active server list and if there has been a change, sends it to the next server. When a new server contacts one on the ring, it is added to the top of the list by the recipient and is sent a copy of the active server list. It updates its active server list, sends the list to the next server and thus the information on the new server quickly propagates round the ring.

If a server fails, the preceding server in the ring will discover the fact when it tries to send on its list. The sender will then remove the failed server from its list and send its list on to the next server. If the failed server had been out of communication because of a temporary network fault it will send its old list to the next server as usual which will treat it as a new server.

A gateway server holds an active server list for each ring to which it belongs. Its domain list could include the domains to which it is a gateway or an entry called "others" as a catchall for unknown domains. For example, in fig. 4.1, the gateway servers in site1 will include sites 2 and 3 as domains for which they are responsible in the active server list circulated within site 1. If this external domain list becomes too large, it could be omitted and the "others" entry will indicate that requests for information about unknown domains on site 1 should be sent to the gateways. Each server know the addresses of all the other servers in its domain (i.e. ring), and hence where to direct requests for information on sub-domains.

There is some similarity beteen our appraoch and that described in [Quirt 87], but their hierarchical structuring is based on physical networks, whereas ours is based on logical domains.

4.3.2 Node - Server Interaction

We will describe the interaction between a user's configuration node and the server by means of an example shown in fig. 4.2. The user's commands are represented in italics.

When a user wishes to perform configuration management he typically runs a configuration node as a normal process on a Unix host system. The configuration node is given the path name to the user's home domain, and a list of possible server locations. It uses the same technique as a new server to locate one of the servers in the ring, and asks for the server responsible for its domain. The node sends a new node status report to the server for its domain, which allocates it a *primary server* (itself) and one or more *back-up servers*. All nodes periodically report status to the primary server. If a node cannot contact its primary server it tries the back up server, which will provide it with a new list of back-up servers. If a back-up server is removed from the system or fails, the primary server sends a new list of back-up servers to its nodes.

```
% iman                    Start up the Interactive Configuration Node,
                          which knows it is in domain "conicgrp/kevin"
                          - user kevin's home domain. iman locates
                          the server for conicgrp.
iman V5.2
iman? domains             Query server to find out what domains are
                          available
      conicgrp            These are typical top-level sub domains:
      BSC3
      Sun                 Sun Unix host workstations
      Vax                 Vax Unix host systems.
      target              target computers for real time applications
iman? manage target       Add target domain to current context
iman? nodes               Query server to find out what nodes are in
                          context.
      conicgrp/kevin:
            iman
            sampler       A node which had been left running.
      target:
            beano         Bootstrap nodes which permit loading of new
                          nodes on targets
            rover
            topper
iman? status rover        Iman obtains address of rover from server
                          and queries it directly
      rover: bootstrap (running)      Node state
      entryports:
            config
            dload
      exitports:
            status
iman? move rover to kevin     Moves a node into domain kevin -
                              prevents its use by other users.
iman? remove target           Remove target domain from context
iman? create testnode at rover Load a new node called "testnode"
                              on target rover
iman? start testnode          Start it running
iman? nodes                   Check that new node exists
      iman
      testnode
      sampler
iman? quit
%
```

Fig. 4.2 Example Configuration Management Session

It should be noted that the information about the nodes is not replicated but partitioned to reduce the information held by any one server. Replication is not required because all the relevant information is generated periodically by the nodes, the servers' information will be complete within a known time after the failure of one or more server processes.

Any of the operations described in section 3.6 can be carried out via the configuration node on either the domains or on the nodes. This may entail locking nodes to prevent other configuration nodes interfering. The configuration manager nodes are trusted nodes in that they validate configuration requests against ACL information they obtain. This could be a potential security risk if someone replaced our confiuration manger node with one of there own, possibly written in C. This would be a very difficult task requiring considerable knowledge about the underlying system, which is not documented. the security risk is commensurate with that of the Unix and Berkely communication support of the hosts. There are a number of possible improvement such as moving the access checking into servers or use of authentication services which we will investigate for more secure environments.

When a node is created using the configuration manager, it can be told its primary and back-up servers when its code is down-loaded, so it does not have to go through the server location phase. Target microcomputers have a simple bootstrap node built into ROM. Putting addresses of servers in ROM was considered inflexible so a slightly different server location protocol is used. This runs on a bare machine and can have access to broadcast communication. It periodically broadcasts a message until the current target domain server provides it with the primary and back-up server addresses.

4.4 Domain Information Maintained by Servers

The servers maintain both *replaceable node data* and *irreplaceable domain data.* The node data is periodically generated by the nodes in the system and includes the node names, status and node's current domain. If this is lost, for any reason, it will be fully replaced within the repeat period. The domain data consists of domain relationships (parent, subdomains, overlapping, reference etc.), as well as the access control list of each domain.

Each primary server has one or more back-up servers which

hold copies of its irreplaceable domain data. Each of these servers keeps a checkpoint and history of changes to the domain data on backing store. The list of back-up servers is also considered irreplaceable data.

Only the primary server need hold information about the nodes within the domain. The nodes know which domains they belong to. If the primary server fails, the nodes then report to the secondary server.

5. CONCLUSIONS

This paper has shown how domains can provide a flexible approach to structuring the configuration and name management of distributed systems. We believe that the concepts described will scale to very large systems because:

i) The number of configuration managers is unlimited and they can be distributed
ii) The name location information which they require can be both partitioned according to the domain structure and replicated for reliability
ii) Domains provide the means of limiting the user's view of a system to a manageable set of components i.e. the information which needs to be held at one place can be limited to a manageable amount.
iii) Disjoint domains can be developed and later integrated using domain references.

The hierarchical domain structuring into subdomains which can be applied to both configuration management and name servers provides information hiding and flexibility. However the concepts of overlapping domains and cross-references to bypass the hierarchical structure are also needed. Large systems are not static but evolve to meet changing requirements, so facilities to dynamically move components between domains and to change the domain relationships are needed

Although access control lists were used in our environment, the concepts could equally be applied to a capability based system as shown in [Robinson 88a]. The Conic system uses the concepts of configuration manager which performs changes to a system, but it would not be difficult to adapt the ideas to system in which nodes performs their own configuration changes e.g. the binding operation in most remote procedure call implementation.

6. ACKNOWLEDGEMENTS

We would like to thank Naranker Dulay, Jeff Kramer, Jeff Magee, Jonathan Moffett and David Robinson who have contributed to the concepts described in this paper. We gratefully acknowledge the support of the SERC ACME Program (Grant GR/E 62394).

7. REFERENCES

Dulay 88 N. Dulay, J. Magee, J. Kramer, M. Sloman, K. Twidle, "Experiences with the Conic Toolkit" in *Experiences with Distributed Systems*, ed. by J. Nehmer, Springer Verlag Lecture Notes in Computer Science No. 309, pp. 189-212, 1988.

Kramer 85 J. Kramer, J. Magee. "Dynamic Configuration for Distributed Systems", *IEEE Trans. Software Eng.* 11:4, April 1985, pp. 424-436.

Kramer 87 J. Kramer, J. Magee, M. Sloman. The Conic Toolkit for Building Distributed Systems, *Proc. IEE Pt. D*, 134:2, March 1987, pp. 73-82.

Magee 87 J. Magee, J. Kramer, M. Sloman, Constructing Distributed Systems in CONIC Imperial College Research Report DOC 87/4, March 1987 to be published by *IEEE Trans. on Software Engineering*.

Magee 88 J. Magee, J. Kramer,"A Model for Change Management", *Workshop on Future Trends of Distributed Computing*, Hong Kong, Sep. 1988, Publ. by IEEE Computer Society.

Robinson 88a D. Robinson, "Domains: A Uniform Approach to Distributed Systems Management", Ph.D. Thesis, Imperial College, March 1988.

Robinson 88b D. Robinson, M. Sloman. "Domains: A New Approach to Distributed Systems Management". *Workshop on Future Trends of Distributed Computing*, Hong Kong, Sep. 1988, Publ. by IEEE Computer Society.

Sloman 87 M. Sloman (ed.) "Distributed Systems Management". In *Issues in LAN Management*, Proc. IFIP TC6.4 Workshop, Berlin, July 1987, ed. by I. Dallas, E. Spratt, North Holland, pp. 15-46.

Rosenkrantz 78 D. J. Rosenkrantz , R.E. Stearns, P.M. Lewis, "Systems Level Concurrency for Distributed Database Systems", *ACM Trans. Database Systems* 3, 2 (June 1978), pp. 178-198.

Quirt 87 A. Quirt, "Reliable Name Service for Hierarchical Campus Networks". In *Issues in LAN Management*, Proc. IFIP TC6.4 Workshop, Berlin, July 1987, ed. by I. Dallas, E. Spratt, North Holland, pp.203-219.