**Title:** Sharing Resources Through Dynamic Communities

**Topic:** C4ISR/C2 Architecture

**Authors:** Kevin Foltz and Coimbatore Chandersekaran

**POC:** Kevin Foltz

**Name of Organization:** Institute for Defense Analyses

**Address:** 4850 Mark Center Drive, Alexandria, VA 22311

**Telephone:** 703-845-6625

**Fax:** 703-845-6848

**Email: kfoltz@ida.org**

# Sharing Resources through Dynamic Communities


# Abstract

Organizing computer network resources in a secure and efficient way is one that has been studied extensively. Many successful implementations of these networks exist today. The problem of interaction between these networks is an evolving problem with fewer theoretical and practical solutions available. The main problems in such collaborations are ease and quickness of setup, functionality, and security.

We examine some proposed models for forming a dynamic community (DC), and propose a new model for building DCs that addresses some of the shortcomings of other models. We propose extensions to current models, such as addition of sensors to a network. We show how to implement such a DC using Microsoft's .NET technology.


# 1 Introduction

Many organizations have computer networks that manage computers, users, data, and services efficiently. These networks enforce organization policy for these elements, allowing users to access resources in prescribed ways and restricting unauthorized access. The problem of interaction between different organizations is more difficult.

We study collaboration among different organizations using the idea of a dynamic community (DC). We consider not just sharing of data, but collaborations requiring tight integration of all elements of the collaborating organizations. These collaborations may be long-term mergers, short-term goal-oriented tasks, or ongoing projects between different organizations. Rapid assembly of collaborating partners is required, so these DC's must be established considerably more quickly and efficiently than setting up an entire new network. However, they must still provide the proper functionality and security measures to ensure safe interaction and sharing of resources.

## *1.1 Collaboration Scenarios*

To better analyze DC models and their attributes, we have constructed several real-world scenarios where dynamic communities would be desirable. We present four of these examples below; each poses a different set of challenges to DC design.

I. **Company Merger** (*Rapid Assembly*) Two large companies agree to merge. Fully integrating their two distinct networks (including establishing trust relationships) would take many months. A DC could serve as the method of uniting the companies during the initial stages of the merger, allowing essential parts of each company to collaborate immediately. Here the quick-creation feature of dynamic communities is essential, and the DC must accommodate large-scale collaboration on a domain level.

II. **Joint Military Operation** (*Data Isolation*) Military strategists of three different governments wish to collaborate temporarily to plan and conduct a small joint operation. However, no military trusts the others' computer networks completely, and each military wants its computers and resources outside of the community to be completely isolated from the collaboration. In this scenario, only portions of each pre-existing static domain wish to become part of the new community, and strict separation between the community and contributing domains is a high priority.

III. **Proprietary Research** (*Data Confinement*) The FDA is evaluating the effects of a new drug. A thorough examination of the drug requires the FDA to cooperate with the pharmaceutical company that developed the drug and also with a research hospital that has conducted experiments on it. Privacy is a great concern in this collaboration, as the pharmaceutical company wishes to divulge as little information about its research as possible, and the hospital must not release any confidential information regarding its experiments. Neither organization wants the government to have any more access to its networks than absolutely necessary. Additionally, the pharmaceutical company expects the proprietary information it contributes to be used only to evaluate the drug; it must not leave the community. This last requirement requires effective information confinement in the dynamic community.

IV. **Government Agency Merger** (*Secure Rapid Assembly*) As a collaboration of intelligence and anti-terrorism departments of several federal agencies, the Department of Homeland Security is a promising model for a dynamic community. The Homeland Security Act states that six departments, each from a separate government agency, should be transferred into the control of the Secretary of Homeland Security, and the Secretary should also have access to terrorist-related information from any U.S. government agency. This scenario differs from the multi-governmental collaboration described above in that speed as well as security is a prime concern. Not only will much of the shared information be extremely sensitive, but the department must also be able to access resources in a timely manner. Additionally, a dynamic community model for this scenario must contain an efficient method of organizing the vast amount of information involved in the collaboration of multiple U.S. government agencies.

## *1.2 Solution Requirements*

The basic vision for a DC is to combine parts of multiple self-contained organizations in a way that allows them to share resources in a highly integrated way. A DC should be able to efficiently handle anywhere from a few principals to thousands of principals, from up to tens of organizations. For this collaboration a set of policies regarding users, computers, applications, services, and their interactions is required. Enforcement of local policies can remain under the control of the contributing organization, but the policy must be autonomous from the contributing organizations.

A DC must be able to manage additions and removals of principals and resources. Removal of users should be prompt, with no residual privileges. Addition of resources should occur rapidly, and they should be assigned to users in a well-defined way. Removal of resources could be dependent on either the DC itself (e.g. Scenario I) or the contributing member (e.g. Scenario II). Different removal policies should be available at DC creation time.

Organizations' computer networks could contain various devices, such as sensors, that would be useful to a DC. Sensors have different attributes and limitations, such as power constraints, that must be considered, making integration more implementation-dependent than with standard computer network resources. A DC must allow for sensors while addressing these differences.

Management and evolution of policies is also an important consideration. In a DC this importance is magnified by the fact that different groups with different policy structures are merging to manage sensitive information under one governing policy. A DC should provide appropriate measures to help balance internal power.

## 1.3 Prior Work

The Department of Defense Goal Security Architecture (DGSA) [1] gives a high-level view of resource sharing over different physical networks. Information is separated into logical and physical domains. These domains have no relation to each other, so a logical domain can span many different physical domains, and physical domains can contain many different logical domains. The EDS [5] model builds on the DGSA. Information is placed within the three axes of entities, domains, and systems and the EDS model describes information flow and restrictions along these axes.

A peer-to-peer (P2P) network is a simple way to share resources. Each member joining the collaboration makes certain resources available to the community. Requests that cannot be handled locally are forwarded to the peer-to-peer network. These networks are easy to manage, since they are distributed and individual members do most of the maintenance themselves. However, there must be a discovery service to allow efficient communication and sharing by the members of the community. Security in such systems is dependent on each individual implementing appropriate measures.

The Secure Virtual Enclave (SVE) model [6] retains the primary features of a P2P DC. Resources remain under the control of the individual member domains. Each "enclave" contains architecture that maintains a list of its members. Members share these lists and use them to authenticate and authorize users from outside their local domain. Figure 2 shows the basic SVE structure, and Figure 3 shows the infrastructure used in user authentication.

In a University of Maryland Model Dynamic Community (MDDC) [4], separate identity and authorization servers from multiple member domains combine for resource access. Credentials are issued locally, but authorization credentials may be transferred from one domain to another. Figure 4 shows the steps taken when a user in one domain accesses a resource in another domain. The MDDC model also includes a threshold cryptographic mechanism for joint administration, and a method for prompt removal of members from the community.

These DC models each address some requirements for a DC, but none fully satisfy them. The DGSA is not feasible in its current state, as there are no practical designs for implementation. The P2P, SVE, and MDDC models do not address confinement, which would be important in Scenario III. They also do not discuss sensors and other devices. The P2P and SVE models offer no internal controls for group-based management.

Some commercial DC models are available, such as eRoom by eRoom, Inc. [7] and Lotus's QuickPlace [8]. These products fulfill many requirements of dynamic communities. They can be created quickly, they support dynamic membership changes, and they provide a common forum for group communication, task management, and document sharing. However, the sharing of complex resources such as databases and applications is not supported, and the applications do not isolate the dynamic community from the domain of the host server. Also, since they are applications they are subject to lower level attacks, which would undermine their security.

## 1.4 Outline

In the following section we present a new model for a DC. We describe its operation and how it meets the DC requirements. In Section 3 we compare this model to prior models. Section 4 discusses a plan for implementation using Microsoft's .NET server. Existing functionality is discussed, and ideas for adding functionality that is lacking are presented. Sections 5 and 6 offer concluding remarks and ideas for future work.

# 2 CADC Proposal

The Centrally Administered DC (CADC) model attempts to meet all the requirements for a DC as discussed in the introduction. Rather than join community members on a P2P basis, this model essentially creates a new domain from the member domains, which allows the community administration to be centralized. In general, this model does not achieve the transparency of the P2P approaches, since users must be conscious of whether they are accessing resources in their own domain or via the DC. However, by encapsulating policy within the DC it offers a functionally richer model and simplifies several aspects of community operation.

It is worthwhile noting the differences between a CADC and a federation. The two are similar, but with some important differences. In a federation members remain whole entities, but a larger governing body encapsulates them and provides higher-level policy. In a CADC the members each contribute a piece of themselves to a new entity, the CADC. The members lose those pieces and the CADC is formed by integrating those pieces. Figure 1 illustrates these differences.

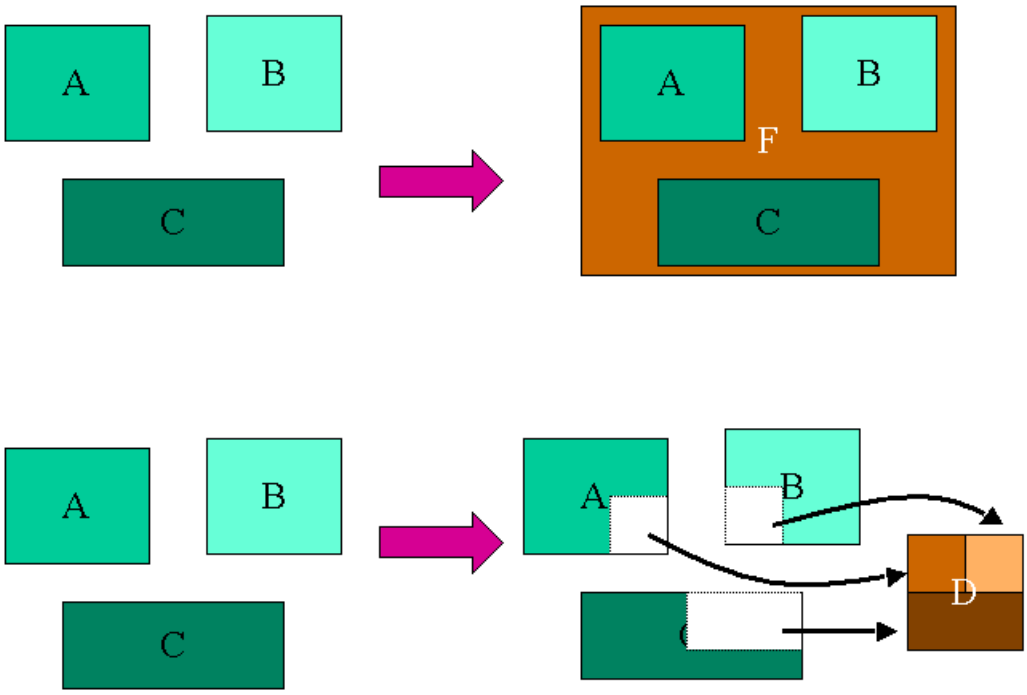


**Figure 1. Top: Creation of Federation (F) from members (A, B, and C). Bottom: Creation of Dynamic Community (D) from members (A, B, and C).**

## *2.1 Basic Operation*

The workings of the CADC are controlled by the central domain administrator (CDA). Users join this CADC much like they would join a static domain; they either submit a request with appropriate information to the CDA—who then accepts or rejects the request based on predefined policies—or the CDA adds users after acquiring the necessary information through offline discussion. Resource access also resembles the familiar static domain process. In a CADC

organizations must transfer their shared resources into the central domain (Figure 1). Switching resources to a new domain increases community functionality and policy flexibility by granting the central domain control over all of the resources.

### 2.1.1 Setup and Configuration

There are several tasks associated with creating a CADC; among these are configuring policy, creating a structure for naming and organizing users and resources, and establishing an authentication mechanism. Fortunately, network operating systems now support automatic configuration for many of these domain creation steps. However, there is virtually no automation in the setting of the hundreds of policies involved in created a CADC. The CDA must tailor the security and user policies to the specific nature of the community, and since these depend on the nature of the CADC's mission, there must be a thorough understanding of this mission and its effect on potential policies. For example, in Scenario I certain information in the DC could be shared freely, since the two companies are likely to benefit from the increased flow of information, and strict security policies could impede progress. However, in Scenario II, where the governments merge briefly and then separate, a policy of strict security would be more appropriate.

The community creator places all information necessary for potential DC members to find, evaluate, and then possibly join the CADC—such as community function, types of resources available, and access requirements and instructions—into a community directory object (CDO). The CDO is published in a well-known location, such as a web page, and users can access the information contained within the object to submit join requests to the CDA. Internal discovery—the method community members use to find out what users and resources are available inside the community—is achieved through an "internal" CDO.

### 2.1.2 Resource and User Management

To share resources to the community, resource owners must first put them in the centralized domain, either by moving or copying them or by joining the server holding the resources to the CADC domain. Resources might remain under the control of their original owners; this policy prevents the CDA from having to manage a large number of community resources. Importing resources is required to provide confinement and give the CDA control over resource policies.

This is especially important in more sensitive communities, where member domains might have strict access and security policies. Coordination of strict individual policies would be considerably more difficult than establishing a working policy as one unit. Additionally, transferring resources eliminates the need to establish a complicated trust relationship between a CADC and its members' original domains. Users and resources can enter the community on an individual basis, and the CDA is not required to trust or communication with other domains.

A CADC handles members entering and leaving the community in much the same manner as a static domain. User entries and departures from the community are handled on an individual basis. An individual enters an already existing CADC by sending a join request to the CDA. That request is evaluated and accepted or rejected by the CDA using some mechanism such as a community-wide vote. If accepted, the new community member is then issued an authentication credential by the CADC, and assigned to the appropriate authorization group. The new user is granted access to specific community resources based on specific member information, the community's security policies, and the decisions of individual resource managers. Policy templates and group policy can aid this process, particularly if a community

accepts multiple users from the same static domain, where similar policies can be applied to each member.

To remove users from a CADC, the CDA simply revokes their identity credentials and removes references from the discovery service. Resource owners are responsible for updating their ACLs; however, this process is not time-critical because former members should be denied all access on the basis of their revoked identity credentials. The fate of resources owned by users leaving a CADC is not specified. In many cases, resource owners will remove their resources when they leave the community (e.g. Scenario II); however, in other cases these resources might be considered essential for community function (e.g. Scenario III). Since the CDA ultimately controls all community resources, it does have the power to prevent a resource owner from removing its resource from the DC. However, with local control a user can delete or modify resources, thus effectively removing them, despite the CDA's wishes.

## 2.2 Sensors

Sensors are a useful addition to many computer networks. For example, law enforcement teams would find sensors useful to monitor communities under their watch. Military coalitions could also use sensors to monitor a combat zone remotely. Sensors are similar to other computer network elements, since they often have memory and processors and can communicate using similar channels and protocols. However, sensors are often distributed in different ways, have different levels of resources available, such as limited battery life, and appear in different numbers and with different functionality than other elements in a network. As a result, we must be prepared to treat sensors and similar devices differently than other network elements.

The operation of a sensor is more constrained than that of a standard computer. Current technology is enabling smaller sensors, with lower power consumption. For these ultra-small and ultra-low-power devices, we must think about networking under a new set of parameters. Instead of speed of network connections and speed of processors, the new primary constraint is often energy (or energy per bit transmitted). With small devices, computing power generally decreases with decreasing size. However, communication power does not necessarily decrease in a similar way.

The Smart Dust project [3] has a method to avoid the communication power problem. A high-power base station sends out light signals, and the sensors respond by adjusting reflectors to either send light back to the source or deflect it away. Thus, little communication energy is required at the sensors, since sending a bit amounts to simply orienting a tiny mirror in one direction or the other.

An easy way to allow communication between sensors and computers is to use standard Internet (TCP/IP) protocols. The problem is that these protocols are not optimized for devices with limited power and changing network topology. Although this implementation would be easy to set up, its performance would suffer due to the mismatch between the protocol and the system on which it is used. To push the limits of integration of the two types of networks requires not just a common interface, but more detailed information about the elements, naming, organizational structure, and interaction policies in each network. Since the sensors have energy constraints, potentially less reliable communication, and specialized functionality, we consider applying a subset of the privileges and responsibilities of normal DC members to the sensors. This has the potential advantages of sensor energy conservation and efficient directory access.

## 2.3 Robust Management

In a typical system, there are critical services that must be provided by different components. Attackers can target any one of these components as a way to compromise services

and ultimately disrupt the entire system. Consensus-based administration (CBA) is a way to distribute authority for important services. When a certain number of the members agree, action is taken, and any number of malicious users (or corrupted users) less than some threshold cannot disrupt the honest members. By requiring the approval of many principals to invoke a service, an attacker must work harder to affect the system's operation. Such systems are also more robust to faults caused by normal accidents or improper configurations. Another application for consensus-based administration is for systems that are inherently distributed and require agreement to perform an action. The mapping here is natural, as CBA allows a single decision to result from multiple individual decisions in a secure way. In this case, CBA aids the system by providing a secure framework in which to make these decisions.

We look at some basic ideas of consensus-based administration. The main goal of consensus-based administration that we will consider is allowing a single point of failure to be distributed among some larger number, $n$, of units in such a way that even if some threshold, $t$, of them are compromised or malicious, the remaining honest units can complete the desired action. The threshold $t$ can be no larger than $(n-1)/2$. Otherwise there could be two groups of size $n/2$ that both have authority to take an action (based on their numbers), but disagree on the action to take. So the honest members must be in the majority in any simple threshold scheme.

A simple way to accomplish consensus-based administration is to combine a number of individual keys using a threshold technique. The $n$ servers create their own individual public key pairs and the set of all the public keys is used as the public key for the threshold scheme. A problem with this approach is that each signature or encryption requires a key of size proportional to $n$. As a result, this method does not scale well. Also, this technique requires changing the signature and encryption algorithms for a system's users, which is not simple for existing systems.

Jarecki [2] discusses Feldman's method to share secrets among $n$ members, such that any $t+1$ honest members will compute a valid common secret. It is based on the (assumed) difficulty of the discrete logarithm problem and the ability to interpolate a $t$-degree polynomial using $t+1$ of its values. Jarecki points out a flaw in this algorithm and proposes a way to fix it. This algorithm can be used to determine access to services in a CADC using CBA. The shared secret in this case is the decision about whether to grant or deny access. Individuals may use agents to make decisions for CBA, and only handle more complicated cases "manually." This speeds the granting of requests for users of services and allows those members involved in CBA to focus on the more important decisions.

# 3 Model Comparison

We compare the different DC models. We look at setting up a DC, how resources are managed in a DC, and how users are managed. Since the P2P, SVE, and MDDC models do not incorporate sensors, we leave out such a comparison.

## *3.1 Setup and Configuration*

Because there is no new domain to configure in a peer-to-peer network, P2P DC creators do not have the numerous policy decisions that a CDA must consider when forming a CADC. Instead, policies are left to the discretion of the individual member domain administrators of the P2P DC. This results in a distributed policy that depends on the policies of the individual contributing members. On the other hand, "member domains" of a CADC have no control over contributed resources except through the CDA. This allows the CADC to be autonomous in setting policies.

Both the SVE and MDDC models base membership on a domain level. Member domain administrators notify the MDDC of members that will join, and interested members in a domain form an enclave that becomes part of an SVE. Domain-based membership is convenient in a company-merging situation; however, on other occasions only a subset of users in a member domain may want to have any relationship with a DC. Achieving this separation between domain members that join and stay out of DC's is problematic in the P2P models. Domains establish proof of identity by issuing identity credentials that each entity in the domain must trust as a condition of membership. However, because there is no community-wide identity CA, P2P communities require that each member domain trusts all identity certificates from all other member domains, which includes certificates for domain members not participating in the P2P DC. This trust represents a potential security vulnerability.

### *3.2 Resource and User Management*

Sharing resources in a P2P DC is a simpler process than in a CADC because shared resources remain with resource owners instead of being transferred to a central domain. This eliminates the need for community administrators to find storage space for resources, negotiate with static domains over resource transfers, or deploy interfaces (such as database clients) for the shared resources.

An additional issue involves the location of the community-wide authorization CA that the MDDC authors recommend for enforcing joint ownership. This CA is supposed to be isolated from each member, but actually achieving that goal is a challenge, as by the nature of a P2P network there is no self-contained location outside of the member domains. If the CA exists on a member domain, then that domain administrator may have more control over the jointly-owned resources than other member domains. In a CADC, the CA is a fundamental part of the DC itself, and it is managed by the CDA.

Both the SVE and MDDC models handle user entry and departure from communities similarly. Prospective members join a community by submitting a request to join and having that request granted; they leave by notifying an administrator. The SVE model does not allow for explicit timely removal from the DC. The MDDC corrects this by ensuring that when a user leaves the DC that user no longer has access to resources of the DC, and no other principals to which the user has delegated authority can use this authority to access resources.

The models differ on how to preserve unique user names. While a CADC forces users to use a separate identity for all community operations, both the MDDC and SVE models make community interaction transparent—users can continue to use their static domain identity while accessing resources in and out of the community. This feature makes users' tasks easier, although it requires that each static domain administrator have knowledge of every DC member to map the names to identities that resource owners in that domain can understand.

The P2P and SVE models do not address CBA. The MDDC model offers some basic mechanisms for allowing group-based decisions to be made. However, the requirements for joint administration proposed include the unanimous consensus of all parties. The CADC model allows different thresholds to be set to allow for single points of failure, or multiple compromised entities.

# 4 Implementation with .NET

We look at implementing a CADC using Microsoft's .NET server Enterprise Edition. The widespread use of Microsoft's Windows 2000-based operating systems is the main motivation for considering .NET for DC's. It is new, but it is largely based on the Windows 2000 architecture. We focus mainly on the aspects in common with Windows 2000, and leave many of

the newer, .NET-specific components for future consideration. The main components we consider from Windows 2000 are Active Directory (AD) and the idea of domain-based computing. Together these offer a good starting point for building a DC.

## 4.1 Basic Implementation

The first major step to set up a DC with .NET is to set up Active Directory (AD), the directory service for the DC. AD provides a way to organize and locate users, computers, sites, domains, trusts, services, and organizational units (OU's). Management of such entities also occurs through AD, allowing addition, deletion, and policy modification. With the directory service established through AD, the DC policies can be established for the elements listed in the directory.

The next important step is establishing an auditing policy, ranging from local computers to domain-wide auditing. Security incidents will occur, and it is important to record appropriate information to allow the determination of their cause and resulting effects. Having auditing encourages non-malicious organizations to join a DC, since it adds some accountability to members' actions. Reckless behavior by one organization is recorded and can be reviewed and penalized if desired.

At this point, more domain controllers can be added that run .NET server and AD. These will all replicate the AD data and provide a higher degree of reliability and performance for AD. Finally, the CA's are established, starting with a root CA and its self-signed certificate, and progressing to other CA's with certificates signed by the root. This takes care of many of the requirements for a DC. However there are requirements that .NET cannot yet suitably handle.

## 4.2 Additional Functionality Requirements

Sensors pose a difficult problem for a DC. Different sensors can have widely differing properties. Also, many sensors are low-power devices that cannot afford any inefficiency in communication or DC membership maintenance. So, we have a large space of possible implementations, each of which has specific and inflexible requirements. We propose two different ways that AD could be extended to allow incorporation of sensors into a DC. The first method of adding sensors is direct insertion into AD. Just as computers and users are maintained in AD, sensors are added as an additional type of entity with their own properties. This method would work well for autonomous sensors with ample CPU cycles, memory, storage, power, and communication bandwidth. A user would have access to basic controls and sensor readouts, as determined by DC policy and sensor functionality. Also, sensors, like users, could be pooled into groups for easier administration.

The idea for the second type of sensor resource is to take a layered approach in adding sensors to a DC. Instead of providing access to sensors directly, we provide access to custodians, sensor organizational units that have the specific task of maintaining sensors, groups of sensors, or other custodians. Users access sensors through these custodians, which then communicate with the sensors directly themselves or indirectly through other custodians. The custodian framework allows efficient scaling to large numbers of sensors, using a tree structure. It also isolates users from the particular implementation of the sensors, allowing sensor changes and upgrades to be transparent to users. However, the result is a reduced integration of the sensors into the DC. For this reason, we would use the layered approach of integration only if direct insertion is infeasible.

For consensus-based administration in AD, we also encounter difficulties in .NET and AD. A CBA implementation in AD should be closely tied to the existing user and group structure of AD to allow reuse of existing functionality. We propose adding CBA to AD by creating a new type of user, a CBA user. This CBA user has a list of members, which must come to consensus before action is taken. A CBA user really represents a group of users, but it has a unique identity

and has permissions and rights assigned to it like an ordinary user. DC resources that require CBA would be placed under the control of the CBA user, which would then request authorization from its member administrators before taking actions. For example, a database service could be placed under a CBA user, and when a request is made to change database policy the CBA user would query the member administrators and take action based on their responses. When a threshold number of administrators vote in favor, the action is taken.

Since there is considerably more overhead in performing actions through a CBA user than through a normal user, due to the many votes that must be collected, it might be best to only assign important tasks to CBA users. However, the use of such CBA users would ultimately be up to the organizations that implement the DC.

# 5 Conclusion

Different organizations, when motivated by a common goal, will wish to collaborate, which requires sharing and integration of their resources. Dynamic community models provide frameworks in which such collaboration can happen quickly and securely. Models for DC's have been proposed, stressing different aspects of collaboration. The DGSA model considers the high-level view and stresses security by restricting the flow of resources. The Peer-to-peer model is a simple sharing scheme in which different domains can join and leave a collaboration and share data with other members of the collaboration.

The SVE model is a more complete DC model. It is similar to the P2P model, except resources to be shared are specifically sectioned off and made available to the SVE. The MDDC model takes this farther by introducing explicit removal of members and resources by the DC and allowing for group administration of important resources. Each of these models solves parts of the collaboration problem, but none of them offers a complete solution. In particular, the addition of sensors is not addressed, and group-based internal control is not well developed.

We propose the CADC, a centralized model that continues the trend away from the distributed sharing of P2P systems and more toward a system for resource integration. This model completely separates resources from their contributing domains, requiring either movement or replication of resources into the CADC domain. Users must have separate identities inside and outside the CADC. This model requires more initial work to set up, but allows tighter integration of resources when established. For implementation, we consider the .NET server, specifically Active Directory. We find that much of the functionality of a DC is available in AD, and we propose ways to add functionality for sensors and consensus-based administration.

# 6 Future Work

For the future, it would be desirable to implement working DC's using .NET and examine which aspects work well and which ones need revision. Since the resources available for sensor and CBA integration are limited, a deeper examination of these issues, involving testing of various integration techniques would be valuable. Much of the work would be based on case studies making use of DC's to solve real-world problems. Initial tests would be for non-critical collaborations without sensitive data, with more realistic and sensitive testing as DC implementation improves.

Work on CBA would also be valuable. There are different methods of achieving CBA, but the main problems DC's will face with CBA are not completely clear yet. For example, the delay of waiting for authorization using CBA could negate the potential gains of CBA. Work in developing agents to make CBA decisions would be good to explore, to examine what portion of decisions could be made immediately and what portion would require longer latencies.

# 7 References

[1] *Department of Defense (DoD) Goal Security Architecture (DGSA)*, Center for Information Systems Security (CISS), Defense Information System Security Program (DISSP), Version 3.0, 30 September 1995.

[2] S. Jarecki, "Efficient Threshold Cryptosystems," Ph.D. Thesis, 2001, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

[3] J. M. Kahn, R. H. Katz, and K. S. J. Pister. "Next Century Challenges: Mobile Networking for Smart Dust." *In Proceedings of MOBICOM*, pp. 271-278, Seattle, 1999.

[4] H. Khurana, "Negotiation and Management of Coalition Resources," Ph.D. Thesis, 2002, Department of Electrical and Computer Engineering, University of Maryland, College Park, MD.

[5] E. Schneider, E. Feustel, R. Ross. 1997. *Assessing DoD Goal Security Architecture (DGSA) Support in Commercially Available Operating Systems and Hardware Platforms*. IDA Paper P-3375, Institute for Defense Analyses, Alexandria, VA.

[6] D. Shands, R. Yee, J. Jacobs, E. J. Sebes, "Secure Virtual Enclaves: Supporting Coalition Use of Distributed Application Technologies," Proceedings of the Network and Distributed Systems Security Symposium, San Diego, February 2000.

[7] eRoom. http://www.eroom.net/eRoomNet/

[8] Lotus QuickPlace. http://lotus.com/products/qplace.nsf

# 8 Appendix

Included here are diagrams illustrating different DC models.

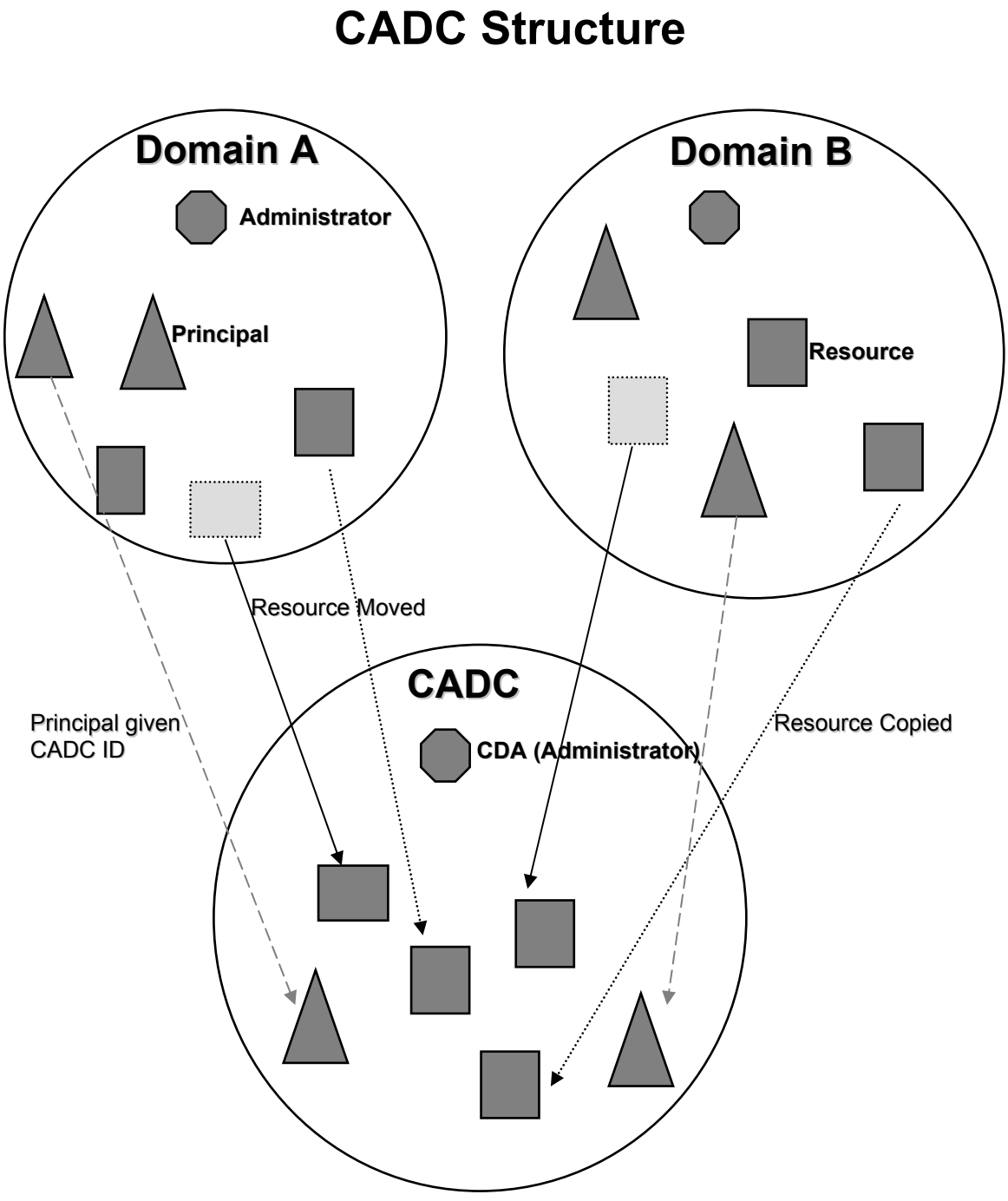


**Figure 1:** Structure of a CADC. Note that all community resources are moved or copied into the community domain, and principals have new identities in the CADC.

# Secure Virtual Enclave

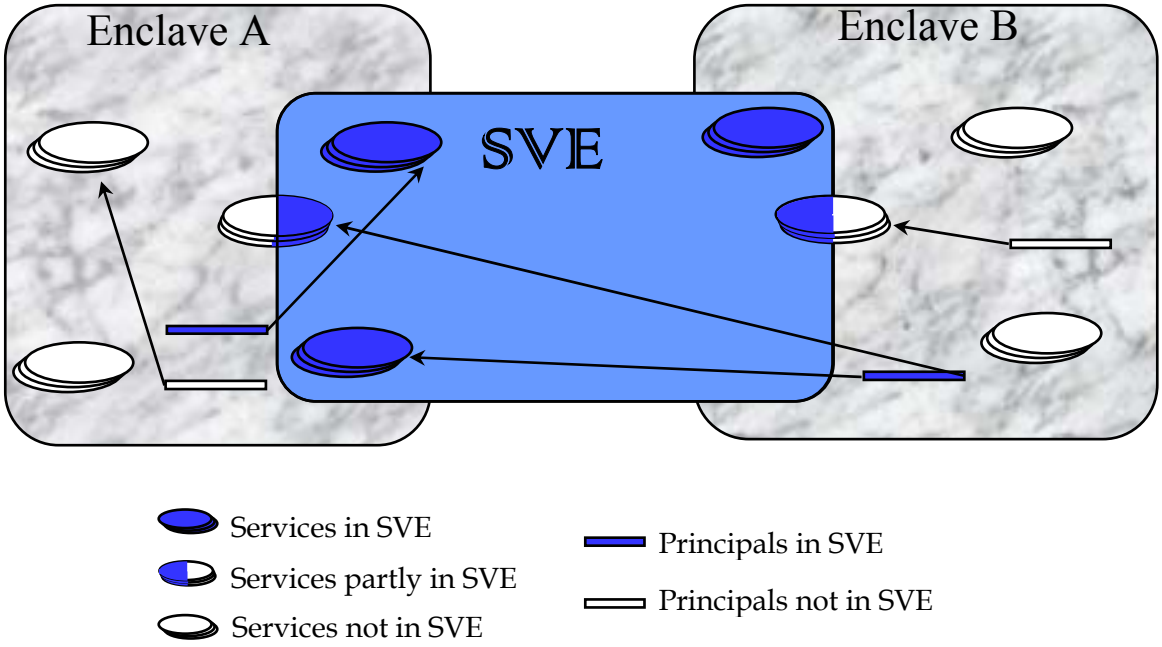


**Figure 2:** Composition of the Secure Virtual Enclave dynamic community model. Note that all community resources are part of an already existing enclave and principals in each enclave can access resources from their enclave that are also part of the SVE.

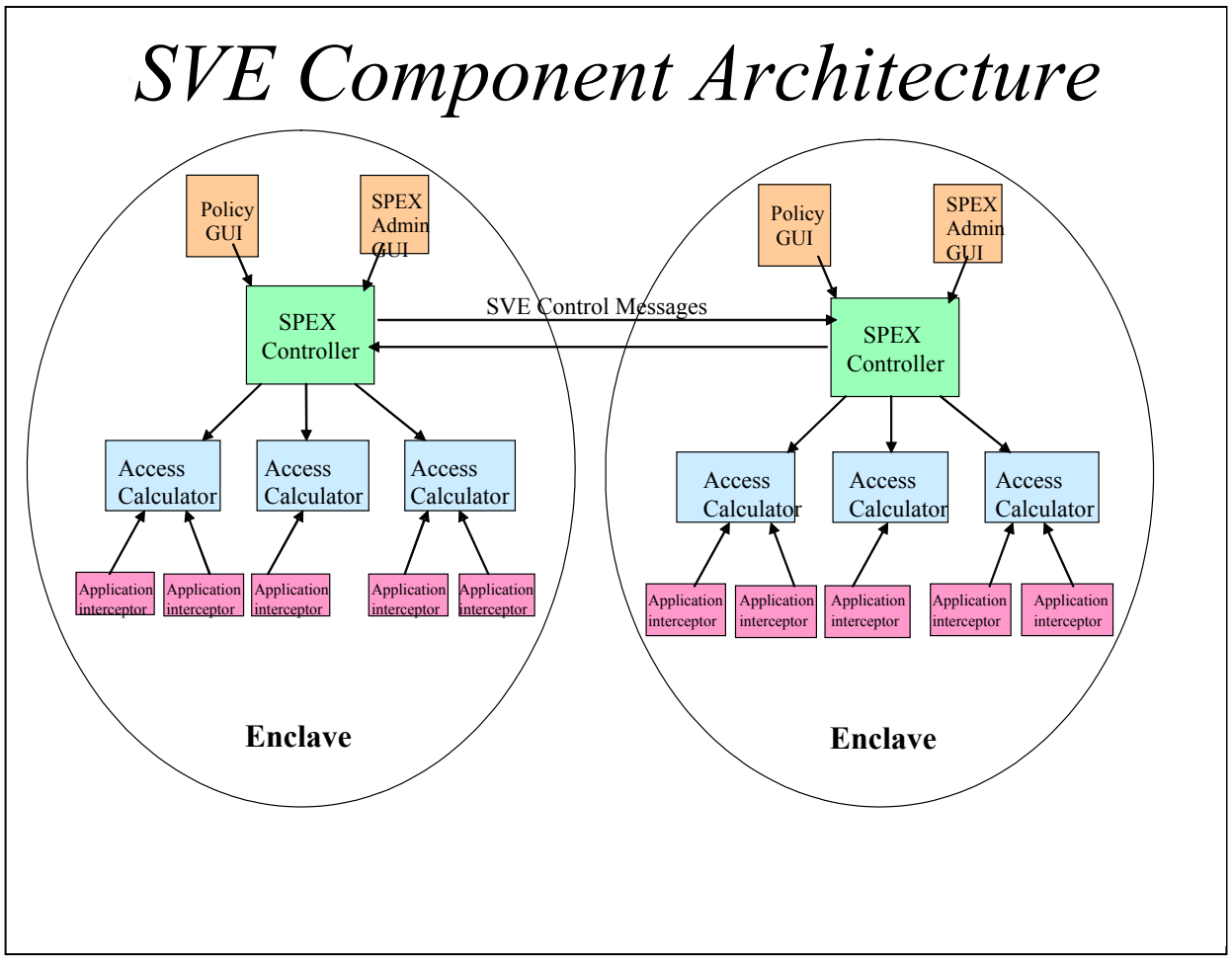


**Figure 3:** Infrastructure used in an SVE. SVE Control messages contain names of community users from each enclave. Access requests to a server in an enclave are intercepted by interceptors, which query the access calculators for a decision on whether to grant the request. Access calculators base the decision on name mappings from the SVE Policy Exchange (SPEX) Controller.

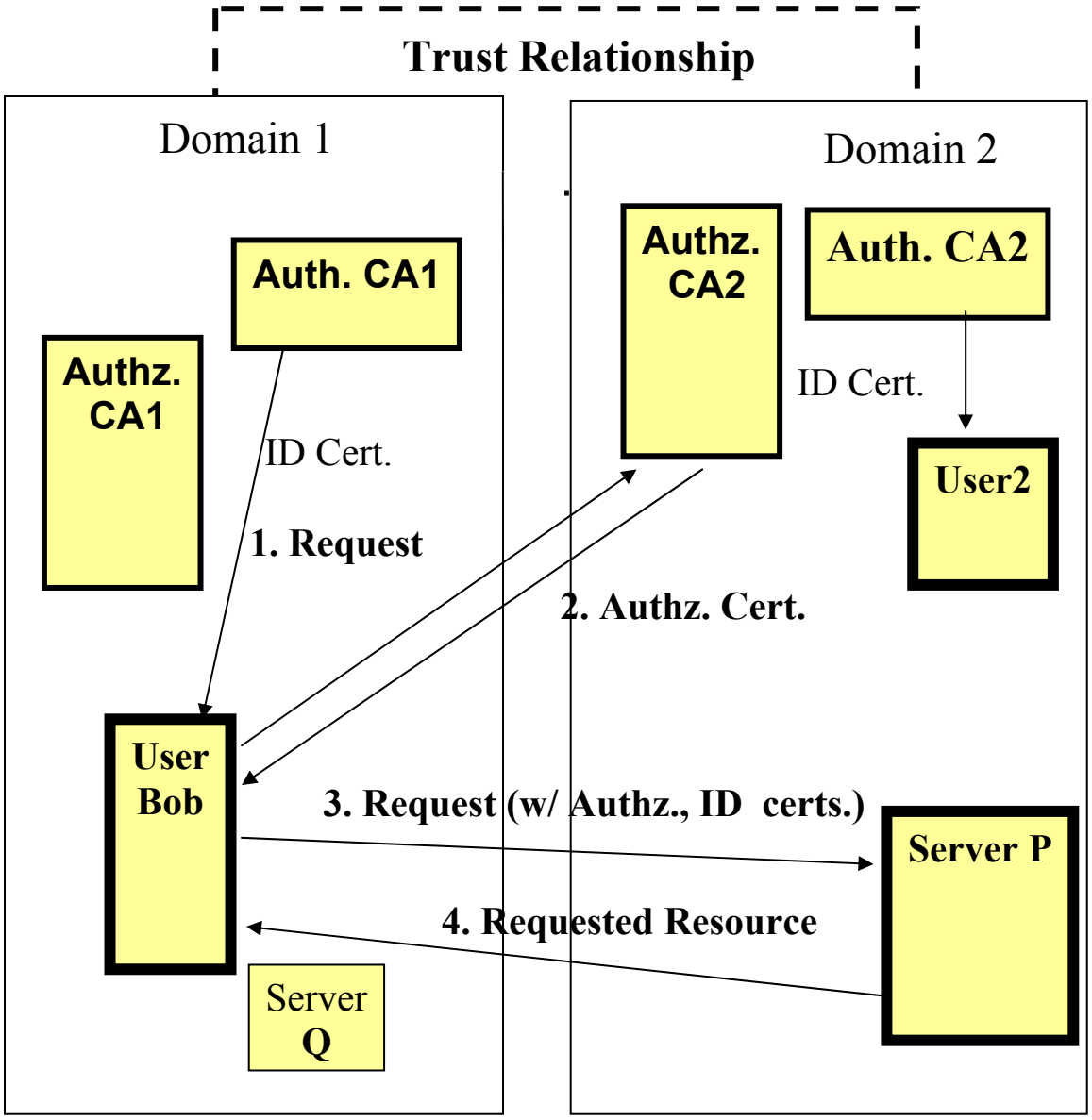


**Figure 4:** Overview of resource access in an MDDC. Bob first requests an authorization credential from the authorization server in P's domain (2). Then Bob sends a request to P with his ID credential and the authorization credential he just received (3). This is sufficient for access (4).