DisCo: A Distribution Infrastructure for Securely Deploying Decomposable Services in Partly Trusted Environments (TR2001-820)

Eric Freudenthal, Edward Keenan, Tracy Pesin,
Lawrence Port, and Vijay Karamcheti
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
{*freudent,woodiek,tracyp,lport,vijayk*}*@cs.nyu.edu*

Abstract

The growing popularity of network-based services and peer-to-peer networks has resulted in situations where components of a distributed application often need to execute in environments that are only partly trusted by the application's owner.

Such deployment into partial or unstable trust environments exacerbates the classical problems of distributing decomposable services: authentication and access control, trust management, secure communication, code distribution and installation, and process rights management. Unfortunately, the application developer's burden of coping with these latter issues often dominates the benefits of service distribution.

The DisCo infrastructure is specifically targeted to the development of systems and services deployed into *coalition environments:* networks of users and hosts administered by multiple authorities with changing trust relationships.

The DisCo infrastructure provides application-neutral support for the classical problems of distributed services, thereby relieving the developer of the burden of independently managing these features. DisCo also includes support for continuously monitoring established connections, enabling corrective action from an application to cope with changing trust relationships.

Our experience with building a secure video distribution service using the DisCo toolkit indicates that the latter permits distributed secure deployment into a partly trusted environment with minimal application developer effort, affording the advantages of natural expression and convenient deployment without compromising on efficiency.

1 Introduction

There is a growing number of applications that deploy systems of distributed components into potentially untrusted network environments. This application model offers a number of benefits: increased efficiency, as in a server application that caches data for mobile clients; increased scalability through caching and edge servers; scalability, better local performance, and failover in peer-to-peer networks.

However, this application paradigm requires that the developer address several problems associated with network environments: distributed authentication, trust management, code distribution, and process rights management. Addressing these problems becomes more difficult when security and trust assumptions can change dynamically as in a *coalition environment.*

In a coalition environment, hosts and users from different organizations are temporarily allied to pursue a common goal. For the duration of the operation, proprietary resources should be shared by coalition members for their mutual benefit. However, this alliance may be dissolved quickly and abruptly, and resources shared under the temporary trust relationships established during the alliance should now be denied to untrusted entities.

Coalition environments present a specific set of challenges in addition to the classic problems of network applications. Sensitive code should not be released to a potentially untrusted host that may later use it against the provider. Instead, a component that may be abandoned should be delivered – a dumb proxy or a restricted implementation with limited functionality, and the choice of an appropriate component is based on environment criteria that can not be known until the time of request. In addition, trusted hosts that have received application components may become untrusted, and the delivery of sensitive data updates must be stopped.

Applications that fit the dynamic coalition environment model include sub-contracted use of proprietary resources, applications for inter-agency cooperation, and networked use of personal resources from public terminals.

Existing techniques and packaged solutions address different subsets of the problem of partly trusted environments, but these solutions are not well-integrated. No one solution addresses the problem of decomposable applications in untrusted environments as a whole. Distributed component technologies like CORBA [11] and J2EE [18] have authorization models that do not respond well to dynamic changes in trust structure, and peer-to-peer protocols like JXTA [15] have not thus far included decomposed, heterogeneous components in its service delivery model.

The DisCo infrastructure is intended to make the work to publish a decomposed, distributed, secure application in an unstable trust environment as easy as construction of a similar application in a completely trusted network environment.

DisCo is built on a set of reusable core application-neutral abstractions that may be implemented by the developer. These include the following:

- The *Activator*, which provides a monolithic interface for the request, installation, and distribution of decomposable network services, forwarding all distribution requests to the services themselves, allowing implementation-specific solutions to the partial trust problem.

- A *Service Distribution* infrastructure which enables service decomposition, dynamic service distribution to support scalability and locality, and mechanisms to allow individual services to determine what level of functionality is returned to a request.

- The *Monitored Subscription* model, a simple pairwise subscription abstraction provided to build indirect connections between clients and servers, which enables dynamic response to changes in trust relationships, update of connection properties, and fault tolerance through automatic re-connection to primary and backup servers.

DisCo also makes use of existing technologies as well as other projects under development by the DisCo group. DisCo uses Public Key Infrastructure for credential validation [14] and dRBAC, a Role-Based Authentication scheme currently being developed by the DisCo group [9]. Service discovery and secure connections across hosts are enabled through the DisCo group's ongoing work on the Switchboard connection infrastructure [10].

The current DisCo implementation has been developed in Java SDK 1.3, using Java's support for remote procedure call (the java.rmi classes). We have based DisCo on a small number of reusable abstractions and have designed the implementation to be extensible to minimize redundant functionality development.

Using DisCo tools and classes, we were able to rapidly develop a video distribution service for use in a dynamic trust environment. With a minimum of development effort, the completed application was endowed with the following sophisticated functionality:

- A request for use of the video service is honored with a component appropriate to the current state of coalition trust relationships.
- A host's copy of the video service has monitored subscription relationships with the application components it has distributed across the network.
- These monitored subscriptions respond to changes in coalition trust relationships by correctly disabling or re-enabling the video feed to the affected users and hosts.
- The video service and user subscriptions can dynamically re-connect to backup servers if broken.

Use of DisCo meant that we needed to write only a minimal amount of video handling-specific code to create the video service, and much of this overhead was required by our choice of an out-of-the-box video framework. We feel that much of this overhead will be reduced by further development of generic data transport functionality.

The rest of this paper is organized as follows. Section 2 presents in more detail the application structure and challenges that are addressed by the DisCo infrastructure. The architecture of DisCo is described in Section 3 and its implementation in Section 4. Evaluation of the infrastructure using a video distribution service is discussed in Section 5. Related work is presented in Section 6 and we conclude in Section 7.

2 Background

The DisCo infrastructure targets the secure deployment of *decomposable distributed applications* in dynamic *coalition environments*. The latter consist of networked hosts spanning multiple administrative domains with asymmetric and dynamically changing trust relationships with one another. In this section, we define decomposable applications, and list the new challenges that must be addressed when deploying them in dynamic partly-trusted settings. To make these concepts more concrete, we also introduce a video distribution application, which serves as a running example throughout this paper.

2.1 Structure of Decomposable Applications

Securely deploying applications in partly trusted environments, particularly when the established trust relationships can change over time, requires that one make some assumptions about application structure. In particular, it should be possible to "give up" on an application component that has been deployed on a host that was initially trusted, if at a subsequent time its trust level decreases.

One application model that can satisfy this requirement is exemplified by several component-based frameworks such as Sun Microsystems' J2EE [18], Object Management Group's CORBA [11], and Microsoft's .NET [12]. Applications in such frameworks are constructed out of multiple, interconnected components. Application deployment corresponds to mapping the components on a distributed network of hosts. An important characteristic of such *decomposable* applications is that the distributed components must *continue to interact* to achieve the application's full potential.

The DisCo infrastructure relies on this last property, using it to manage application mappings in dynamic environments. In particular, two kinds of application structures lend themselves to secure deployment. The first class includes restricted components with limited functionality that may be abandoned on a host that can no longer be trusted. Examples include proxy implementations that forward sensitive queries to a trusted host, providing useful solutions to queries without compromising the security of the problem-solving code.

The second class of applications derives its security from the interactions that flow between the various components. For such applications, the underlying system can cope with unpredictable changes in trust simply by controlling whether or not a particular component is allowed to interact with the other components. Examples of such applications include those whose components generate, process, and forward data streams, such as an information dissemination service. In this case, updates on the data stream can be stopped if a previously trusted host were to become untrusted.

Note that proprietary commercial applications that are increasingly being offered to clients on a pay-per-use or subscription basis, or with an expiration period (e.g., virus checking software), have the same kind of distribution and trust concerns.

2.2 Deploying Decomposable Applications in Dynamic Environments

When deploying decomposable applications in coalition environments, one needs to address not only the classical problems of distribution (such as secure connectivity, node and link heterogeneity, and fault management), but also concerns that are particular to the dynamically changing nature of trust.

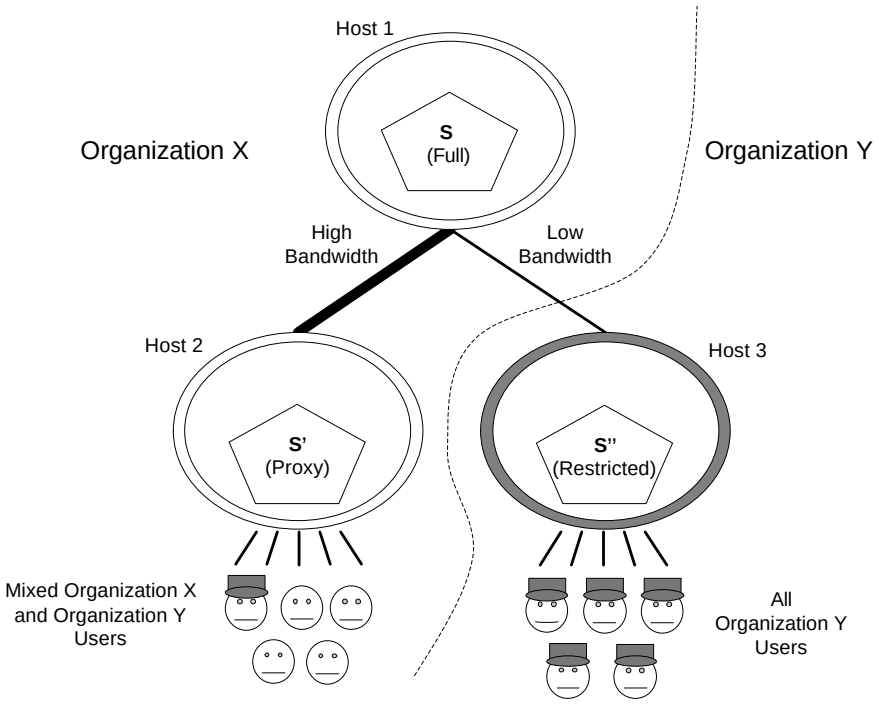

Figure 1: Deployment of a decomposable application in a coalition environment of hosts from multiple organizations. In addition to the classical concerns of distribution (e.g. bandwidth differences between the Host 2 connection and the Host 3 connection with Host 1), the deployment needs to take into consideration that hosts and users may be trusted to different extents. In this figure, the deployment handles this dynamic trust situation by distributing a proxy to Host 2 and a restricted implementation to Host 3.

Figure 1 highlights these new concerns that are described in additional detail below:

- Application components must be *specialized* for installation in different trust situations. Since hosts may be trusted to different degrees, and these relationships are unstable, there must be a mechanism

for generating components appropriate to each level of trust. In Figure 1, the component that gets mapped to Host 3 must take into consideration the fact that Organization Y's trust relationship with Organization X can change.

- Application components may need to be *downloaded on demand* from the network. Given that the trust levels of a host chosen for deployment (say the nature of requesters on Host 2) cannot be known a priori, a component delivery agent must be able to dynamically assess the requesting host's credentials and return appropriately specialized components.

- Connections between application components must be subject to *constant monitoring*. In a dynamic trust environment, relationships between components (e.g., between S and S" in the figure) may need to be broken or revised abruptly. All pairwise relationships should be monitored and appropriate corrective action taken as soon as possible.

- Application deployment is driven by *explicit validation of trust* among otherwise anonymous entities. Since in settings involving multiple administrative domains, there is unlikely to be agreement on a single trust management authority, there needs to be a decentralized mechanism to verify credentials and their implications.

The DisCo infrastructure presents an integrated approach for handling both these new concerns as well as the traditional concerns of distribution, specifically coping with node and link heterogeneity and fault tolerance.

2.3 Example: Video Distribution Service

As an example of the kinds of applications the DisCo infrastructure enables, consider the need for a secure video feed between networks of hosts in a temporary coalition, say troops from multiple countries collaborating on a military mission. For the moment, the goals of the members of the coalition are the same, but they have not been in the past, and they will most likely not be in the future. Giving every member of the coalition access to one member's proprietary video feed will help with the short-term shared goal, but the owner of the video feed will not want to share it once the current problem is solved, and everyone's goals diverge again.

Data is transmitted via point-to-point-transmissions in a dynamically deployed distribution tree. A trust authority is established to control distribution and consumption of the video feed. It specifies mechanisms for trust management for both the distribution and consumption relationships.

Since hosts may not have all required components to support video applications, the code must be provided and installed on demand across network connections. The distribution of the necessary code and data may be restricted to trusted hosts, and the agents distributing this information must also be credentialed to perform those tasks.

A host that is running a video distribution service receives a video feed, which it can then serve to other hosts or provide to individual users on the host, in any approved combination. A distribution agent must continuously monitor that its consumers (whether hosts or end-users) are authorized to perform their tasks.

Updates to this trust structure (in both internal and external domains) should result appropriately in individual hosts being added to or cut out of the point-cast network and individual users being granted or denied access the video feed.

We will return to this example later in Section 5.

3 DisCo Architecture

3.1 Overview

The DisCo architecture is informed by the specific challenges of deployment of a distributed, decomposed application into a partly trusted dynamic coalition environment. Figure 2 shows the overall architecture of the DisCo infrastructure. The infrastructure relies upon a core set of modules: Activators, Service Distribution mechanisms, and Monitored Subscriptions, which interact with an externally provided auxiliary set of components: service discovery, role-based trust management infrastructure, and PKI. Development of these auxiliary components is being worked on in other parts of the DisCo project.

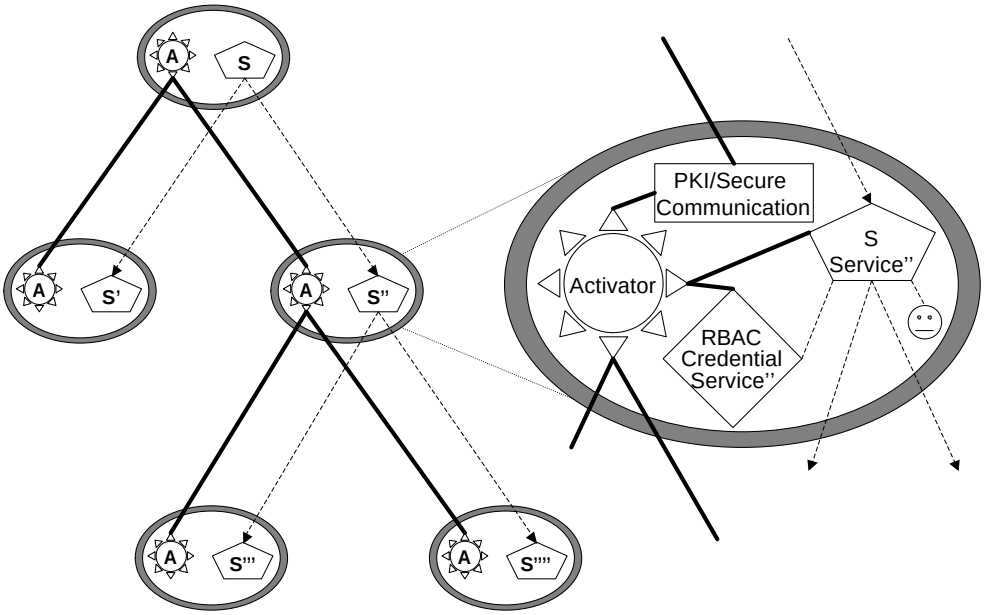

Figure 2: Overview of the DisCo Architecture, showing a network of DisCo nodes

- The *Activator* manages the set of services installed on a host. It helps bootstrap service connectivity. The Activator handles inter-host requests for services and forwards them to the appropriate local service installations. The Activator installs code returned by service requests across network hosts.
- *Service Distribution* is the mechanism for providing hosts and users with application components that they request. All DisCo services contain the basic machinery for self-distribution. Individual service implementations are able to handle requests as their function requires, returning code installations for specialized tasks appropriate to the requester's trust level.
- *Monitored Subscriptions* help provide security, dynamic behavior, and fault tolerance. A pairwise interactive relationship between two objects can be a Monitored Subscription, enabling this relationship to be as responsive to environment changes as is desired. This includes changes to trust levels, changes to network service quality, and changes to server load. As soon as a change in the environment can be detected (whether it is an operator-induced change such as permission revocation or a non-deterministic change such as network hardware failure), the DisCo Monitored Subscription functionality can rapidly disseminate the information across the network of hosts.

Auxiliary components used by DisCo include the following:

- *Service Discovery* can be provided by a secure network communication abstraction such as Switchboard [10], currently under development by the DisCo group.
- *RBAC* provides a scalable model for distribution of user and host trust levels in a coalition environment. Each member of a coalition will be able to maintain a separate trust hierarchy, constructing and removing bridges from other trust hierarchies as desired. The DisCo group's dRBAC system [9] is being developed to support delegation chains maintained across multiple hosts.
- The existing *Public Key Infrastructure* is used to prove the identity of objects communicating across the network.

3.2 Core Components

3.2.1 Monitored Subscriptions

To maintain integrity and coherence in a decomposed and distributed application the DisCo Infrastructure provides a subscribed relationship template. Monitored Subscriptions are the fundamental abstraction upon which most DisCo functionality is implemented.

Any two objects which have a sustained interactive relationship implement a pairwise subscription mechanism that enables regular delivery of updates from publisher to subscriber, two-way communication, liveness checks, and automatic cleanup when the relationship ends.

The server-side subscription monitor maintains a close relationship with the local RBAC mechanism. Thus when the trust environment changes and a coalition member revokes the credential upon which the Monitored Subscription is based, the Monitored Subscription can be stopped immediately.

Monitored subscriptions are implemented for both local relationships and for remote relationships using RPC. The subscriber object in the relationship may cache secondary subscriptions which they can use to dynamically re-connect if the connection to its publisher object is lost.

3.2.2 Activator

The Activator coordinates and manages the services installed on a host. Hosts and users that need to use a service make their requests to the Activator. Services which need to interact with other services on a host ask the Activator for a handle to the needed service. A host need only be deployed with an Activator, and the Activator obtains services from other known hosts and codebases.

The Activator handles all the components of requests for service: the needed service ID, the credentials of the user or host making the request, the proof of the requester's identity. The Activator takes these components and routes them to the appropriate service. The Activator handles the returned instance as well, passing it back to the original requester.

If a host fails to find a needed service locally, it will make a call to another known Activator, appending its host credentials and forwarding the request using the same semantic as the original request. The result of this forwarding mechanism is that a request for a service will propagate through a tree of Activators until it reaches a host that can fulfill the request. Appropriate implementations are passed back through the network of requesting hosts. This results in a wider distribution of the application components, creating more hosts which are able to fulfill requests for service and giving the application better load balance and redundancy across the network.

3.2.3 Service Distribution

Services can be distributed across a DisCo network through the system of chained Activator-handled requests described above. They can also be instantiated through explicit class loading from both local and remote codebases.

In Activator-chained requests, individual service implementations may determine the level of functionality returned to a request for distribution. The service evaluates the request based on service-specific set of criteria (RBAC trust relationships, network locality, guarantee of secure communication, quality of service, etc.) and returns a decomposed service component appropriate to the request: a full copy of the service, a proxy implementation, a restricted version of the service with limited functionality, etc. Thus the component considered best for the job is provided, and sensitive components are not distributed to potentially untrusted hosts.

Codebases are used to bootstrap a service. The Activator has functionality to allow a user to explicitly load a service from a codebase and create a root instance. Access to these codebases is authorized using the same RBAC model that determines dynamic service distribution.

3.3 Auxiliary Components

3.3.1 PKI

Individual users and hosts make requests for resources across potentially untrusted networks. In this environment, there must be a scalable means of verifying the identity of the requester. All DisCo hosts, users, and services have identities that can be verified using message signatures generated with existing Public Key Infrastructure [14].

DisCo can use a PKI and secure communication abstraction like the DisCo Switchboard [10] to handle these requirements.

3.3.2 RBAC Trust Management

Because the DisCo Infrastructure is intended to be used in dynamic coalition environments with multiple trust authorities, DisCo is built with an eye toward integration with a distributed RBAC trust management system. RBAC is particularly suited to the scaling and separate trust hierarchy challenges of a dynamic coalition.

The DisCo Group's dRBAC [9] is being developed to function in dynamic coalitions and provides the credential management that the DisCo infrastructure needs.

3.3.3 Service Discovery

Service discovery is a problem that can again be handled by a third-party communications abstraction like the DisCo Switchboard [10]. Activators maintain a registry of installed resources that can be consulted to create service advertisements and network maps.

3.4 Scalability

The DisCo system is built to be highly scalable. There is no notion of a central Activator that could create a bottleneck. Services may originate and propagate from any Activator on the network. Subscriptions between service instances can be dynamically broken and reconnected to take advantage of changes to network topology. Members of a coalition may individually set policies for access its resources, without having to rely on the potentially bottlenecked services of a universally trusted third party.

3.5 Lifetime Description

Propagation of a service can begin with a user requesting access to a particular service not currently installed on a host. The DisCo Infrastructure provides request routing, request validation, and automatic establishment and cutoff of Monitored Subscriptions as seen in Figure 3.

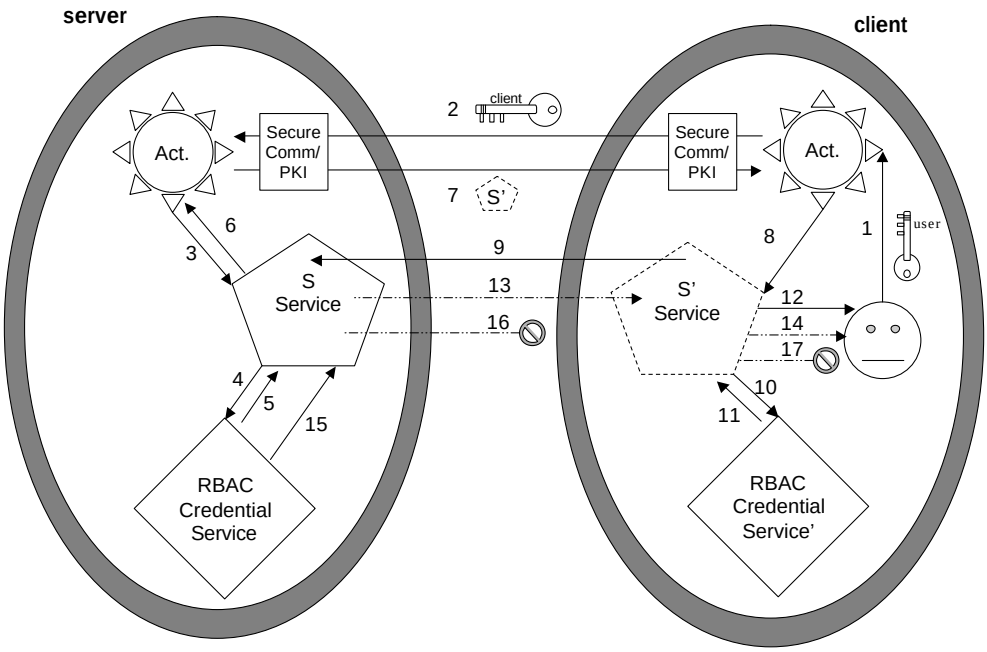

Figure 3: Lifetime of a DisCo user request for service, including request propagation (1-5), download and service instantiation (6-9), client/user interaction (10-12), and Monitored Subscription maintenance (13-17)

1-5. *Request propagation:* A user on a host requests access to a service S, providing credentials and proof of identity (1). The local Activator does not have an installed copy of S in its registry, and so it appends its own host credentials and proof of identity to the request and forwards it to another network Activator, known through the communications and service discovery abstraction (2). The server Activator receives and authenticates the request and forwards it to the local copy of service S (3). Service S forwards the client host credentials to a local RBAC Credential Service for authorization (4). The RBAC Credential Service is contracted to watch the client's Monitored Subscription to S. It returns the mechanism that will stop the subscription should the RBAC Credential Service discover that the client's credentials have become invalid (5).

6-9. *Download and service instantiation:* S returns to the Activator an appropriate application component: a full copy, a proxy, or a limited implementation of S (6). The new service component S' is serialized and passed via RPC to the client Activator (7). The client Activator installs S' on the local system, logging it in its registry so that it may distribute it to other hosts and local users (8). Upon installation, S' immediately contacts S on the server host, establishing the Monitored Subscription (9).

10-12. *Client/user interaction:* S' is given the original user request for access and passes it to the local RBAC Credential Service for evaluation (10). The RBAC Credential Service is contracted to watch the user's Monitored Subscription to S'. It returns the mechanism that will stop the subscription should

the RBAC Credential Service discover that the user's credentials have become invalid (11). The user is now a subscriber to S' (12).

13-17. *Monitored Subscription maintenance:* S provides S' with a stream of subscription updates (13). S' provides the user with a stream of subscription updates it receives from S (14). The server host's RBAC Credential Service learns that the client host's credentials are no longer valid (the coalition has been broken), and it immediately notifies S to stop the Monitored Subscription to S' (15). S stops the stream of updates to S' (16). S' no longer has updates to provide the user (17).

4 DisCo Implementation

4.1 Code Structure

The core DisCo components are relatively small, yet the reusable abstractions they offer are very powerful. The basic concept of *indirection* is used throughout DisCo to implement mechanisms that can respond to dynamic trust relationships.

The core components currently stand at approximately 1700 lines of code. They interact with auxiliary components comprised of approximately 2000 additional lines. The small size of this code reflects the efficacy and reusability of our core set of abstractions.

We have also designed the DisCo infrastructure so that a developer can rapidly create service objects and subscription indirection objects through simple extension of the DisCo base classes, which already contain all needed mechanisms for Monitored Subscriptions.

Given Java's current support for remote procedure call (the java.rmi classes), Java seemed the logical choice for development of the DisCo infrastructure. Java SDK 1.3 currently provides limited language-provided mechanisms for secure class loading, we have opted to provide DisCo functionality for explicit class loading through codebases. Java SDK 1.4 promises useful tools for dynamic and distributed secure class loading, which we will use to replace the current need for codebases. These language advances will allow us to model seamlessly integrated service and code distribution.

4.2 Component Details

Figure 4 shows component interactions, illustrating the objects used to maintain Monitored Subscriptions.

4.2.1 Monitored Subscriptions

The DisCo Monitored Subscription is the primary abstraction used to maintain application integrity in a partly trusted dynamic coalition environment. The Monitored Subscription allows there to be a level of indirection between an application component and a potentially untrusted user or host. The provided component contains no sensitive functionality that could not be abandoned and which will be useless without the subscription update stream supporting it.

The use of indirection in Monitored Subscriptions allows additional beneficial features for use in a dynamic coalition environment. Interfaces between objects in a Monitored Subscription as shown in Figure 5 may be remapped for application-specific access control. Rate control on user requests can be regulated, allowing users a limited number of requests per unit time, thus protecting the server application installation from a Denial of Service attack. Upgrades to client access can be made by modification of the indirection objects.

A Monitored Subscription is maintained between two objects: the server object that implements the DisCo *Publisher* interface and the client object that extends the DisCo *Subscriber* class. The pairwise

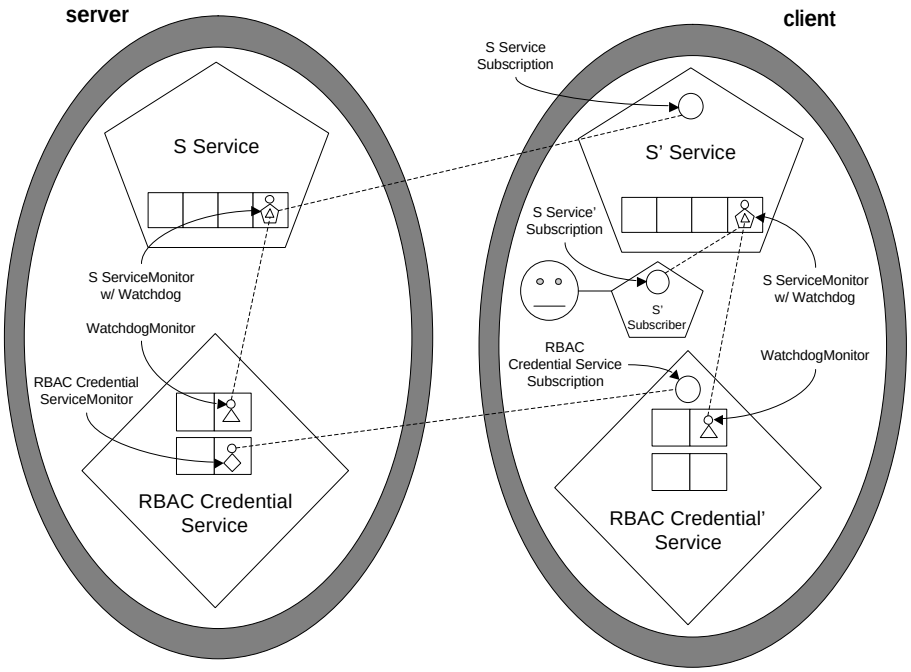

Figure 4: DisCo Infrastructure component interactions. Monitored Subscription relationships are shown in dashed lines.

subscription mechanism itself is comprised of a *SubscriptionMonitor* object held by the Publisher and a *Subscription* object held by the Subscriber.

The *Publisher* interface provides methods for requesting a subscription relationship, as well as hooks into the *PublisherModule* class which maintains a collection of SubscriptionMonitors.

The *Subscription* object embodies the subscription provided to the client. The Subscription object has a one-to-one relationship with a SubscriptionMonitor, but a cache of Subscription objects means that the client is not without a service provider if the connection to its current server is lost.

The *Subscriber* class, extended to be a client, maintains collections of Subscription objects (gathered via network connections maintained by network resource managers such as the DisCo Switchboard [10]). When the active Subscription becomes invalid, a new Subscription can be drawn from the cache of available Subscriptions and started. The cache of Subscriptions can be ordered and updated for validity and preference based on connection criteria: load balance, trust, quality of service, etc.

SubscriptionMonitor objects are the server-side mechanism of the subscribed relationship. The SubscriptionMonitor provides update streams of data and functionality, two-way communication, liveness checks, and load monitoring. SubscriptionMonitors provide a level of indirection between sensitive Service components and potentially untrusted clients.

The *SubscriptionMonitorWatched* object is an extension of SubscriptionMonitor that includes functionality to stop the subscription should the trust relationship on which the subscription is based become invalid.

A SubscriptionMonitorWatched object is itself the client in a subscribed relationship with the agent that authorizes the subscription it monitors. When a SubscriptionMonitorWatched object is created, it contracts with a local Credential service to watch for any change in the authorization state of the subscription.

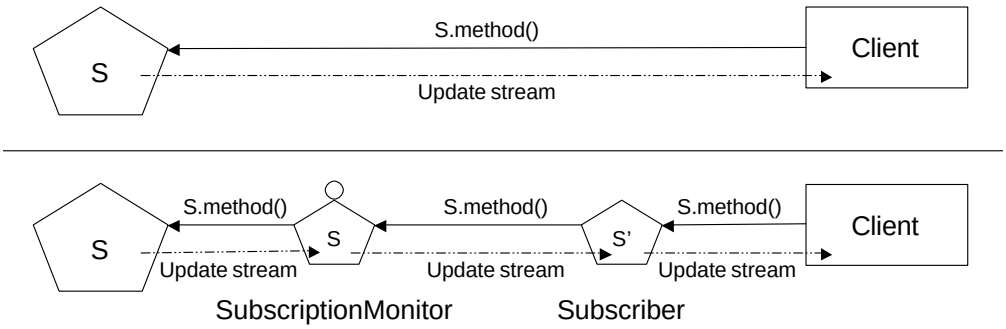

Figure 5: Indirection objects in DisCo Monitored Subscriptions. At top is a traditional client-server relationship, in which the client makes direct method calls on the server, and the server provides the client directly with updates. At bottom is a DisCo Monitored Subscription, in which the server is removed from a potentially untrusted client by a SubscriptionMonitor and the provided application component S'. The interface to S' is identical to that of S, and so the client's use of S' is standard. However, implementation of the S interface on indirection objects is customized to trust and security concerns.

The SubscriptionMonitorWatched does this by requesting a *Watchdog* object from the local Credential service and giving the Watchdog rights to turn the subscription on and off.

The Watchdog is a subscribed client to a *WatchdogMonitor*, held by a PublisherModule in the local Credential Service. A Watchdog is the implementation of the Credential Service's contract with a local service S to watch the validity of a Monitored Subscription to S.

Whenever the Credential Service becomes aware of a relevant change to the trust environment, it re-evaluates the credentials upon which the local Monitored Subscriptions are based. If authorization of the subscription must be revoked, the Credential service informs the WatchdogMonitor, which updates the Watchdog, which in turn cuts off the stream of updates from the SubscriptionMonitorWatched to the Subscription.

The Monitored Subscription implementation gives us a level of indirection between sensitive application components and potentially untrusted hosts and users. The close subscribed relationship between the Credential Service and the server-side control mechanism for Monitored Subscriptions gives us timely response to dynamic changes in coalition environment trust levels. Implementing the Watchdog-WatchdogMonitor relationship by extending the Subscriber-SubscriptionMonitor relationship gives us the reliable connectivity of Monitored Subscriptions, as well as the possibility of using remote Credential Services if one is not available locally.

4.2.2 Activator

The Activator is an application-neutral generic model for service lookup. Activators provide a level of indirection between partly trusted requesting entities and application component providers. Activators manage locally installed services, mediate communication between users and services, link services to helper services, and forwards requests for dynamic distribution of services to other host Activators. The Activator mediates requests for application components but allows the individual service implementations to return an application component appropriate to the request.

The Java interface to an Activator includes methods for users, services and other Activators to call.

4.2.3 Service Distribution

Service distribution across hosts is mediated by communication between Activators, but the individual service implementations determine the level of service functionality returned to a user request (a proxy, full, or restricted instance of the service).

Through the Activator, users call `subscribe()` on a Service implementation to receive a subscription indirection object that will allow them to interact with a service. This method requires that the user provide credentials under the same semantic as a host request for a service: the user's access to the service may be granted or denied due to a change in credential state.

5 Case Study: Video Distribution Service

5.1 Implementing the Video Distribution Service using DisCo

We were able to use the DisCo infrastructure to rapidly build a video and distribution service as well as DisCo-compatible video viewing applications.

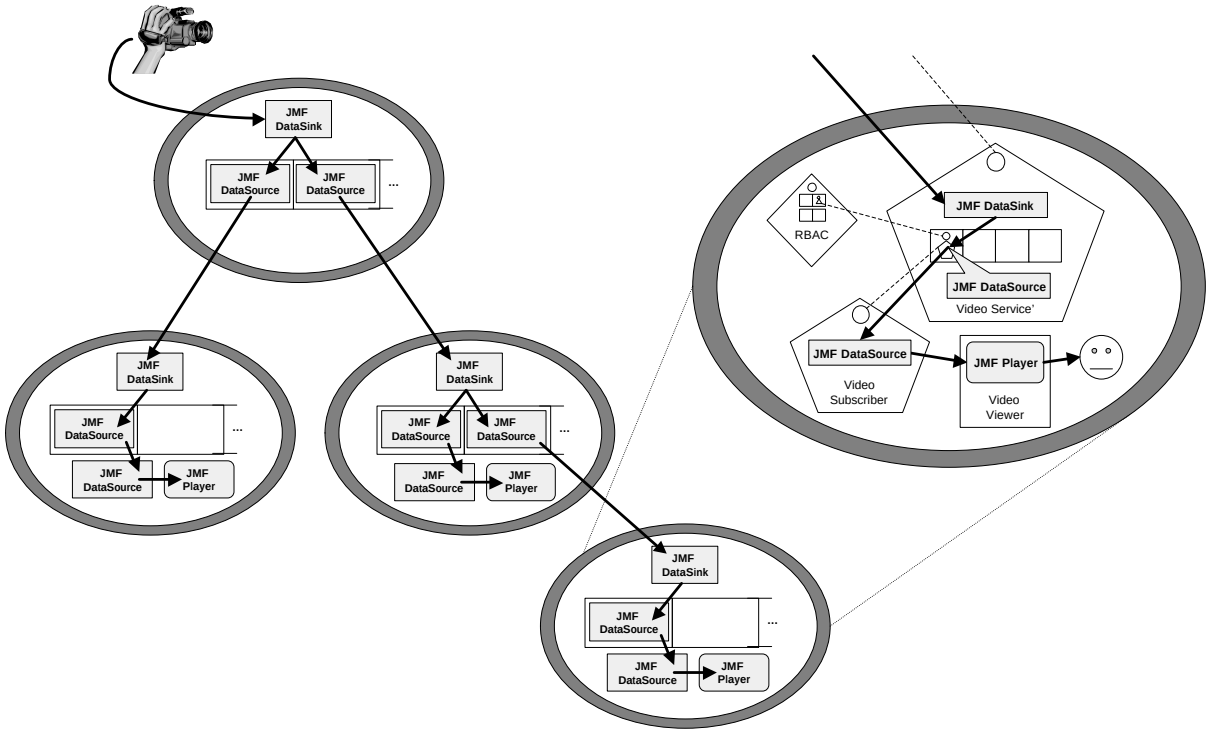

Figure 6: DisCo implementation of a Video Service

By extending the DisCo base classes and integrating components of the Java Media Framework, we were able to create a decomposed video service that could be dynamically distributed and whose access control could be maintained across a network of hosts.

Figure 6 shows the overall structure of the video distribution service, the deployment of JMF components across hosts as well as the integration of JMF components into a decomposed service built by extending DisCo components.

The VideoService implementation extends the DisCo Service class. It contains a JMF DataSink to allow a multiplex video stream. The VideoServiceMonitor extends ServiceMonitorWatched to allow credentialing of the video feeds to other VideoService instances and users. The VideoServiceMonitor contains a JMF DataSource that is linked to the containing VideoService's JMF DataSink.

Users can view the video feed with a VideoViewer application. VideoViewer applications also extend the DisCo Service class, meaning that if it were not installed locally, the Activator could request a copy of VideoViewer from another Activator and receive it using standard DisCo service distribution mechanisms.

A successfully started VideoViewer requests a VideoSubscription from the local VideoService. The VideoService returns a VideoSubscription whose JMF DataSource is a clone of its VideoSubscriptionMonitor JMF DataSource. The VideoViewer contains a JMF Player that receives its video from the VideoSubscription's JMF DataSource.

5.2 What does Disco enable?

The Video Service created using the DisCo Infrastructure has the following capabilities:

1. The Video Service can self-propagate over a network of DisCo-enabled hosts. Instances of the VideoService were dynamically downloaded and installed by explicit user request and by indirect request when the user asks the Activator for a VideoViewer which in turn asks the Activator for a VideoService.

2. The components of the Video Service have monitored subscription relationships with other instances of the Video Service on the network, as well as with the user applications that interact with the service. Subscriptions between hosts could be invalidated, stopping the video feed to the client VideoService and to all users and hosts subscribed to the client VideoService as well. Individual user subscriptions to a VideoService could also be invalidated, stopping only their video feeds and no other. These video feeds could all be restarted by re-validating the subscriptions in the exact same manner as they were invalidated.

3. The subscriptions to the VideoService could dynamically re-connect if broken. Halting the VideoService instance on a server host in a subscribed relationship caused the client to request re-connection to a third host also offering VideoService subscriptions. If the request was accepted, the video feed was restored to the client.

4. The VideoService could be released into a network of semi-trusted hosts with the guarantee that changes in access rights for users and hosts are propagated quickly. Use of an RBAC credentialing service allowed the video feeds to hosts and users that shared an authorized role to be started and stopped on different hosts throughout the network.

5.3 Benefits and Costs of Using the DisCo Infrastructure

The VideoService implementation code is currently 275 lines. Only 30 lines approximately were needed to integrate the JMF functionality into DisCo service decomposition and delivery mechanisms.

To implement the same distributed video service without the DisCo Infrastructure, the code would have needed to include at least the 1700 lines of functionality provided by the DisCo infrastructure as well as the 2000 lines of third-party functionality that the DisCo Infrastructure has already successfully integrated.

Observable inefficiency in the system was due to RPC data copy operations (distributing the service instances between hosts over network connections) and latency in JMF connectivity. The video was fed as fast as JMF allows.

There is no noticable in-line cost due to subscription monitoring (except at state changes when RBAC credential information was propagated throughout the system), which in genneral involves two indirect function calls. Otherwise, the cost is dominated by copy of the service components across host boundaries at the time a service instance is installed.

5.4 Summary

The DisCo Infrastructure allowed the rapid development of a decomposed and distributed video service application and did not introduce any significant overhead in the delivery of the video.

The cost of writing this service with all the features necessary for deployment into a dynamic coalition environment was dominated by the cost of implementing the same network application without any of the code distribution and trust management features that DisCo offers.

6 Related Work

In its broadest sense, DisCo is related to many technologies that enable secure use of applications in partly trusted environments. There are several aspects of this problem ranging from service discovery protocols [19], establishing secure connections between clients and servers [14, 8, 20, 4], to various access control [17] and trust management approaches [1, 16, 5], to mechanisms that protect an execution host by "sandboxing" potentially malicious application components.

Rather than discuss how DisCo relates to each of these technologies, we restrict our attention to three broad categories of approaches: *distributed component technologies* such as J2EE [18], .NET [12], and CORBA [11], recently proposed *peer-to-peer computing* infrastructures such as JXTA [15], and wide-area grid technologies such as Globus [7] and Legion [13]. DisCo can be thought of as blending the objectives of the first two classes of systems: it derives its decomposable services model from the first, and its distribution protocols and trust considerations from the second.

Distributed Component Technologies Both DisCo and distributed component technologies share the view that an application is built up from a set of interconnected components, which may be deployed on different hosts in a distributed system. Like DisCo, these technologies provide "container" environments, which offload most of the programming burden associated with operating in a distributed environment. All of J2EE, .NET, and CORBA have an associated security model, permitting their use in partly trusted environments.

Despite these similarities, there are three important differences. The primary difference stems from DisCo's monitored subscription model, specifically its ability to continually monitor the state of object pair interactions and take corrective action in an application-specific fashion whenever necessary.

Second, DisCo differs in its support for component specialization: component technologies do not offer any explicit support for adapting an application to networks with different, dynamically changing resource and trust characteristics.

Finally, DisCo's general trust-management model marks a different design point from the traditional authorization model used in such frameworks. For instance, the J2EE authorization model of mapping client user identities to groups and groups to roles which have specific method calling rights cannot react to dynamic changes in trust relationships.

Peer-to-Peer Infrastructures Recently, there has been a great deal of interest in general infrastructures for peer-to-peer applications. Such efforts, exemplified by the JXTA approach [15], focus on the construction of peer-to-peer applications that rely upon the distribution of homogeneous objects and services among a large

number of peers. JXTA technology development thus far has emphasized locality and trust challenges [2], and has not dealt with decomposable applications.

DisCo's activator tree structure can benefit from the hash-based service lookup protocols proposed in infrastructures such as JXTA [15] or Chord [3], and its support for on-demand service specialization can naturally accommodate locality considerations. The latter also provides DisCo with an advantage: a DisCo service can customize and dynamically download to a client a service component instance that is best suited to local application needs and the client's network environment.

Grid Technologies Technologies such as Globus [7] and Legion [13] share some of the same concerns as DisCo, particularly in their attempts to employ resources belonging to diverse administrative domains into a unified, virtual pooled resource that can be used by all parties. To date, grid technologies have focused more on providing a unified view of the resources and have not focused as much on concerns of dynamic service decomposition and changing trust.

Given the relatively static nature of their target applications and environments, e.g., running a parallel application across multiple supercomputer sites spread across the nation, grid technologies have been able to rely on simpler application partitioning and security models than proposed in this paper. For example, the Globus security infrastructure [6] provides support for mutual authentication, delegation, and single sign-on, after which it allows the application components to connect with each other. However, unlike DisCo, there is no support to monitor that these trust relationships remain valid for the duration of the interaction, nor any mechanism to allow for corrective action.

7 Conclusion

The DisCo infrastructure is a unified approach to the multiple challenges presented by the development of decomposed, distributed applications in a dynamic coalition environment. There is a growing need for this kind of application, and no single technology available currently has a well-integrated approach to the problem.

DisCo Infrastructure applications can maintain coherence and cohesion even as network topology and trust levels shift beneath the application. DisCo Monitored Subscriptions keep the components of the application in constant contact, allowing for rapid update propagation, dynamic re-connection, load balance, and automatic failover.

The DisCo infrastructure has been developed to integrate with trust management technologies suitable to dynamic coalitions such as RBAC, meaning that authorization to services will be executed properly even over distributed network topology.

The DisCo Infrastructure has a low footprint on a DisCo-enabled host, requiring only the DisCo Activator for initial deployment, with services dynamically distributed as needed. These features do not come at an efficiency cost. The DisCo Infrastructure does not measurably degrade the performance of a network application.

The DisCo Infrastructure will continue to become more tightly integrated with auxiliary technologies and reduce the development time and overhead necessary to create applications that perform well in dynamic coalition environments.

8 Acknowledgements

It is impossible to untangle the contributions of Oliver Kennedy, Vladimir Vanyukov, and Jordan Applebaum who regularly attended the DisCo group meetings and constructed prototypes of system components. Joshua

Rosenblatt explored mechanisms for extending RMI semantics to support secure code loading and proposed the techniques to be utilized with Java version 1.4.

This research was sponsored by DARPA agreements F30602-99-1-0157, N66001-00-1-8920, and N66001-01-1-8929; by NSF grants CAREER:CCR-9876128 and CCR-9988176; and Microsoft. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of DARPA, Rome Labs, SPAWAR SYSCEN, or the U.S. Government.

References

[1] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote Trust Management System, Version 2. IETF Request for Comments 2704, Available at `http://www.ietf.org/rfc/rfc2704.txt`, 1999.

[2] Rita Chen and William Yeager. Poblano: A Distributed Trust Model for Peer-to-Peer Networks. Sun Microsystems, Inc. White Paper, Available at `http://www.jxta.org/project/www/docs/trust.pdf`, 2001.

[3] Frank Dabek, Emma Brunskill, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, and Hari Balakrishnan. Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service. In *Proceedings of ACM SIGCOMM*, 2001.

[4] T. Dierks and C. Allen. The TLS Protocol, Version 1.0. IETF Request for Comments 2246, Available at `http://www.ietf.org/rfc/rfc2246.txt`, 1999.

[5] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian M. Thomas, and Tatu Ylonen. SPKI Certificate Theory. IETF Request for Comments 2693, Available at `http://www.ietf.org/rfc/rfc2693.txt`, 1998.

[6] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A Security Architecture for Computational Grids. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, pages 83–92, 1998.

[7] Ian Foster and Carl Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[8] A. Freier, P. Karlton, and P. Kocher. The SSL Protocol, Version 3.0. Internet Draft <draft-freier-ssl-version3-02.txt>, Available at `http://www.netscape.com/eng/ssl3`, 1996.

[9] Eric Freudenthal, Tracy Pesin, Lawrence Port, Edward Keenan, and Vijay Karamcheti. dRBAC: Distributed Role-Based Access Control for Dynamic Coalition Environments (TR2001-819). Technical report, Department of Computer Science, New York University, 2001.

[10] Eric Freudenthal, Lawrence Port, Edward Keenan, Tracy Pesin, and Vijay Karamcheti. Credentialed Secure Communication Switchboards (TR2001-821). Technical report, Department of Computer Science, New York University, 2001.

[11] Object Management Group. Common Object Request Broker Architecture (CORBA) Specification Version 2.5. Available at `http://www.omg.org/technology/documents/formal/corba_iiop.htm`, 2001.

[12] Microsoft Corporation. Microsoft .NET Framework SDK Beta 2. Available at `http://www.microsoft.com/net`, 2001.

[13] A. Natarajan, M. Humphrey, and A. Grimshaw. Grids: Harnessing Geographically-Separated Resources in a Multi-Organizational Context. In *Proceedings of the High-Performance Computing Symposium*, 2001.

[14] R. Perlman. An overview of PKI trust models. *IEEE Network*, 13(6):38–43, 1999.

[15] Project JXTA. JXTA Version 1.0 Protocols Specification. Available at `http://spec.jxta.org`, 2001.

[16] Ronald L. Rivest and Butler Lampson. SDSI – A simple distributed security infrastructure. In *Proceedings of CRYPTO'96*, 1996.

[17] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control: A multi-dimensional view. In *10th Annual Computer Security Applications Conference*, pages 54–62, 1994.

[18] Sun Microsystems, Inc. Java$^{\mathrm{TM}}$ 2 Platform, Enterprise Edition Specification, Version 1.3. Available at `http://java.sun.com/j2ee/docs.html`, July 2001.

[19] Jim Waldo. The Jini architecture for network-centric computing. *Communications of the ACM*, 42(7):76–82, 1999.

[20] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH Protocol Architecture. Internet Draft <draft-ietf-secsh-architecture-09.txt>, Available at `http://www.ssh.com/tech`, 2001.