An Architecture for Distributed *OASIS* Services

John H. Hine[1,*], Walt Yao[2,**], Jean Bacon[2], and Ken Moody[2]

[1] School of Mathematical and Computing Sciences
Victoria University of Wellington
[2] Computer Laboratory
University of Cambridge

Abstract. Role based access control promises a more flexible form of access control for distributed systems. Rather than basing access solely on the identity of a principal the decision also takes into account the roles that the principal currently holds. We present a distributed architecture that supports the *OASIS* role based access control model. The *OASIS* model is based on certificates held by the client and validated by credential records held by servers. We wish to replicate and distribute the credential records to support high availability and reduce latency for certificate validation. Protocols are presented for maintaining replicated credential databases and coping with both server and network failures.

1 Introduction

Role based access control promises a more flexible form of access control for distributed systems. Rather than basing access solely on the identity of a principal the decision also takes into account the roles that the principal currently holds. This set of roles can change dynamically. Indeed the identity of the principal and the roles held can be thought of as constituting a protection domain [6]. Proponents of role based access control also argue that a formal representation of roles can significantly improve the management of access control policies [9, 2, 11].

The concept that a principal should hold access control rights dates from the development of capabilities [10]. The key problem with roles or capabilities when compared with access control lists is the management of the distributed representation of access control rights. To date there have been few designs for role based access control that have adequately addressed this issue. In [2] the authors describe a limited cgi-script based system for an organisational web server. In [3] and [4] the authors describe an Open Architecture for Secure Interworking

* The work described here was undertaken while John H. Hine was a visiting research fellow in the Computer Laboratory, Cambridge University. The work was supported by the U.K. Engineering and Physical Sciences Research Council, grant no. GR/M37592.

** Walt Yao was supported by the U.K. Engineering and Physical Sciences Research Council, grant no. GR/M75686.

Services, *OASIS*, a proposal for a more general design of a role based access control system.

In this paper we present a distributed architecture for the *OASIS* service. The distributed architecture increases availability and reduces latency. Both features are critical to the successful deployment of role based access control into real systems. We present a protocol using weak consistency that is sufficient for maintaining the integrity of the access control policies; also a protocol for recovering from a server crash or network partition is presented.

The following two sections provide an overview of *OASIS*, its major components and operation. We then present the design of a distributed architecture to support *OASIS* services. This is followed by consideration of the implications of server failure and network partition faults. We conclude with a summary of additional work that remains to be undertaken.

2 Policy, Roles, Certificates and Credentials

A claimed advantage of role based access control is the management of access control policy. *OASIS* includes a Role Definition Language (RDL) for the representation of policy. RDL supports the formal specification of the requirements for all aspects of role membership: entry, retention and revocation. RDL specified policy can be translated into a compact form for interpretation by *OASIS* servers. A full discussion of RDL is beyond the scope of this paper. See [3] and [4] for details.

For our purposes it suffices to note that access control policy specifies the necessary requirements for:

1. The entry of a principal into a role and continued possession of that role.
2. The allocation, by one principal to another, of entry to a role and the subsequent revocation of that role.
3. A principal's use of a role to access a service.

A principal requests entry to a role by presenting an *OASIS* server with the prerequisite credentials. If the server is satisfied it grants the principal entry to the requested role by returning a role membership certificate (RMC). The RMC validates membership in the role and may subsequently be presented to an *OASIS* aware service as part of the access control process.

It is important to note that the access control policy specifies both preconditions for role entry and conditions which must remain true for a role to remain valid. For example, it may be required for the principal *Susan* to hold the role DOCTOR-ON-DUTY in order to be admitted to the role WARD-CHARGE-DOCTOR. We would also expect that for *Susan* to continue in the role WARD-CHARGE-DOCTOR she must also retain the role DOCTOR-ON-DUTY. The formal nature of RDL allows *OASIS* to represent these dependencies as a proof tree ensuring that if a certificate becomes invalid other certificates depending on it will also be invalidated. For example if *Susan* loses the role DOCTOR-ON-DUTY her certificate for role WARD-CHARGE-DOCTOR would also become invalid. Role membership certificates may be parameterised. We would expect the certificate for WARD-CHARGE-DOCTOR to be parameterised by the ward identifier.

OASIS also enables a principal holding a suitable role to request auxiliary credentials that can be passed to a third party. In requesting an auxiliary credential the principal may specify pre-conditions, such as roles held, for its use. These are then built into an *auxiliary credential certificate* (ACC), taking advantage of *OASIS*'s existing functionality. The principal may also introduce an arbitrary decision process in deciding which other principals will be passed the ACC. Such a credential may be used by the recipient to obtain entry to additional roles provided the specified pre-conditions are met.

We will use an example to demonstrate the flexibility of auxiliary credentials. Some authority such as a hospital or government health service might issue *Susan* with an auxiliary credential, *doctor*, asserting her medical qualifications. The auxiliary credential certificate would be valid only when used in conjunction with some authenticated principal representing *Susan*, for example at a workstation whose reader held her personal ID-card. Once issued the ACC would be retained across sessions and so allow the DOCTOR-ON-DUTY role to be entered whenever *Susan* logged in.

These auxiliary credentials, which can be used for traditional delegation amongst other purposes, were referred to as *delegation certificates* in [4]. Specific to each ACC the issuer retains a *revocation certificate*, which allows the credential to be revoked at any time. The mechanism allows an appropriate authority to issue and withdraw credentials that control role entry on the basis of alternative decision making processes, possibly external to the computer based system.

3 Overview of *OASIS*

In this section we look at how a principal interacts with a single *OASIS* secured service and how that service manages its role certificates and access control. A *principal* is a process or thread acting on behalf of a particular user or organisation. We assume that each principal has been issued with a unique identifier by an underlying operating system.

Figure 1 shows a single principal interacting with a single service that is secured with *OASIS* role based access control. Role entry and access control are separate functions. As the role entry function issues each role membership certificate a *credential record* supporting the RMC is stored for subsequent use by the access control function.

It is not necessary that the components of Fig. 1 be combined within a single application. For example, an *OASIS aware service* has only the access control function and the secured service. It relies on a separate *OASIS* service to validate certificates presented to it. At the other end of the spectrum an *OASIS* issuing service has responsibility only for managing the access control policy by performing the role entry function, storing credential records and validating certificates for various access control functions. Frequently, such an *OASIS* issuing

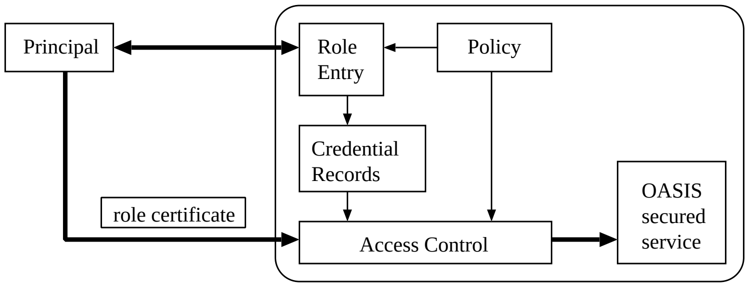

Fig. 1. An *OASIS* service and client.

service will be set up to meet the specialized requirements of an application domain in which access to particular functions will be controlled on the basis of the roles established.

3.1 Role Entry

In most cases a policy will require that an applicant for entry to a role already holds one or more other roles. The applicant must present appropriate certificates to prove membership of these roles.

Each principal must be able to obtain its first certificate(s) without presenting other certificates. Like Kerberos [13] we assume the existence of some form of initial login server. The user interacts with the login server using a password, swipecard or similar technology to authenticate the user's initial principal. Once authenticated that principal is allowed to enter the role LOGGED-IN-USER and is granted a *LoggedInUser* certificate which may be used to enter further roles.

The role entry function will need to verify the principal and certificates as described below. After the server has confirmed that a certificate is valid and current the role entry function is able to issue a new membership certificate for the requested role.

3.2 Certificates

Figure 2 shows the format of a role membership certificate. The first field identifies the role for which this RMC grants membership. This is followed by a set of N parameters for this role.

The next field is a certificate identifier (CID) that uniquely identifies this certificate. The CID has two components, an identifier of the issuing service and a unique identifier for the certificate within the service. The CID may be used to create an audit trail of access control decisions. The next field is a credential record reference (CRR). The combined CID and CRR fields are used by an access control mechanism to locate the credential record held by the service that

role	N	arg1	....	argN	CID	CRR	signature

Fig. 2. A role membership certificate.

issued the certificate. The CID identifies the issuing service and certificate and the CRR provides a hint to locate the credential record.

The final field is a digital signature. Each certificate is held by a principal and may be subject to theft, malicious modification or fabrication. This is guarded against using a hashing function such as MD5 to sign the certificate [14]. The issuing *OASIS* service uses a secret known only to itself in hashing the certificate to produce the signature. The fields that are hashed include the identity of the principal. When the certificate is presented for use it is returned to the issuer for verification. This is done by repeating the hash function and comparing the result with the signature.

3.3 Credential Records

When an *OASIS* server issues a role membership certificate or an auxiliary credential certificate it creates and stores a matching *credential record*. The credential record identifies the certificate and holds state relating to its validity.

The structure of stored credential records establishes a proof tree for each RMC and ACC. Figure 3 demonstrates with a simple example. In this example we assume that *doctor* is an auxiliary credential that has been issued to *Susan*. Assume that entry to the role DOCTOR-ON-DUTY requires both this auxiliary credential and the role LOGGED-IN-USER. *Susan* requests the role DOCTOR-ON-DUTY by presenting her ACC for *doctor* and her RMC for LOGGED-IN-USER. This enables the role entry function and the servers holding the existing credential records to create the proof tree of Fig. 3. A pointer to the new credential record is included with the credential records corresponding to the credential *doctor* and the role LOGGED-IN-USER. (Note that both of these certificates would be parameterised with a persistent identifier for the doctor.)

An acyclic credential record graph is created with some of its links pointing from the credential record database of one service into the credential record database of another. To revoke one of its certificates a service locates the corresponding credential record and flags the certificate as invalid; it then finds links to the credential records of all immediate dependants. These certificates can then also be revoked. The process continues, recursively collapsing a sub-tree of the credential record database.

3.4 Role Use

A principal uses a role by presenting one or more role membership certificates to the access control mechanism when requesting service. Since the certificate

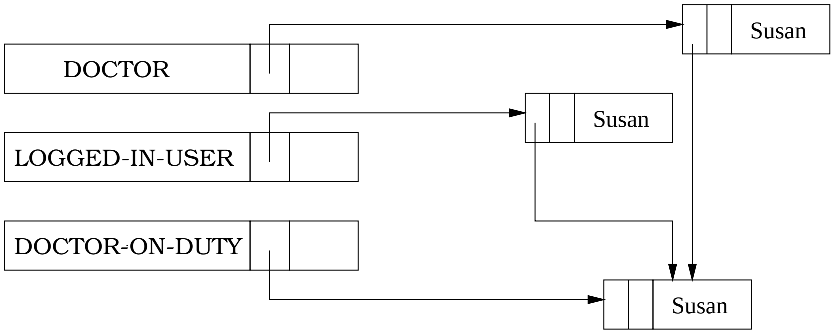

Fig. 3. The certificates and credentials held by the principal *Susan* on the left and their credential tree.

has been held by the principal the access control mechanism must make the following checks:

1. Authenticate the identity of the principal presenting the certificate.
2. Verify that the certificate has not been tampered with.
3. Verify that the certificate has not been revoked and that all necessary conditions for its use remain true.

The principal may be authenticated using a conventional authentication service [8]. The process is completed by referring the certificate and the principal's identity to the *OASIS* service that issued the certificate. The service is identified from the CID. Each service will only recognize certificates issued by services which are explicitly identified within its access control policy. The issuing server recomputes the digital signature using the principal identity supplied. If the signature is correct it uses the CRR to locate the credential record for the certificate in order to ensure that it has not been revoked.

Once the certificate and principal have been verified the access control mechanism uses the identity of the principal, the certified role, and any parameters included in the certificate to determine whether or not to grant the access requested.

3.5 Auxiliary Credentials

An important aspect of *OASIS* role based access control is the ability of one principal to use auxiliary credentials to control another principal's ability to enter a role. Policy for role entry expressed in RDL can require the presentation of an auxiliary credential certificate in addition to one or more role membership certificates. The ACC details additional RMCs required together with any constraints on their parameters. It can be used to extend the role entry policy stored at the *OASIS* server.

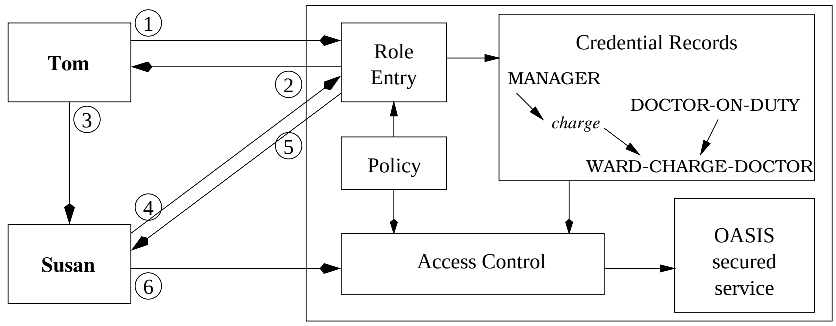

Fig. 4. Delegation in an *OASIS* service.

Let us develop the example of Fig. 3. It is possible that a hospital manager would appoint charge doctors for the various wards. This could be done by supplying an auxiliary credential, *charge*, to the doctor, *Susan*, and requiring that she present the *charge* ACC along with her DOCTOR-ON-DUTY RMC in order to enter the role WARD-CHARGE-DOCTOR.

OASIS manages auxiliary credentials with two additional classes of certificates, the auxiliary credential certificate and a revocation certificate. Figure 4 shows the basic steps that would be used in our example. The six messages are:

1. Principal *Tom* requests entry to a role, MANAGER, that can distribute auxiliary credential certificates *charge*. *Tom* would need to satisfy the policy requirements to enter MANAGER. *Tom* is issued with a role membership certificate for MANAGER and a credential record is stored.
2. While *Tom* holds the role MANAGER he may request ACCs, *charge*, for entering role WARD-CHARGE-DOCTOR. In requesting the ACC *Tom* may specify pre-conditions for use of the ACC to enter the role. These include roles and constraints on parameters of those roles. In our example, the ACC *charge* would require an accompanying RMC for DOCTOR-ON-DUTY and that the principals named in the ACC and the RMC were identical to the presenter. These conditions are embedded within the auxiliary certificate and protected by the digital signature. These conditions are in addition to any imposed by the role entry policy for the role WARD-CHARGE-DOCTOR.
   When certificate *charge* is issued *Tom* also receives a matching revocation certificate, RVK. A credential record for the auxiliary credential certificate is stored. There is no credential record for RVK.
3. Following a negotiation which is not shown, *Tom* allocates the right to enter role WARD-CHARGE-DOCTOR to *Susan* by sending her the ACC, *charge*. *Tom* retains the revocation certificate, RVK. The "negotiation" between *Tom* and *Susan* allows arbitrary information such as references or past experience to be made a pre-condition for entering role WARD-CHARGE-DOCTOR.
4. *Susan* presents the ACC *charge* and the other prerequisites requesting entry to role WARD-CHARGE-DOCTOR.

5. Assuming entry to role WARD-CHARGE-DOCTOR is granted an appropriate RMC, *WardChargeDoctor* is returned to *Susan*. A credential record for this RMC is stored such that it is dependent on the credential record for the auxiliary certificate *charge*.
6. *Susan* presents the RMC *WardChargeDoctor* requesting an operation from the service.

The auxiliary credential certificate contains the usual reference (CID plus CRR) to its own credential record. The revocation certificate contains two references, the first to the credential record for the ACC *charge*, and the second to the possibly parameterised role, MANAGER, under which it was issued.

The revocation certificate may now be used to revoke the allocated role. It must be presented with an RMC for the role MANAGER. The CRR for the auxiliary certificate *charge* is used to perform the revocation by invalidating the auxiliary certificate. Any RMC *WardChargeDoctor* that depends on the auxiliary certificate will also be revoked.

Using auxiliary certificates it is possible to extend policy expressed in RDL. First, the issuer can specify roles and constraints on those roles that must be met by any principal using the ACC, and in that way make use of the *OASIS* certificate validation and role entry functions. Secondly, before passing on the auxiliary certificate the issuer may base the decision on arbitrary logic, including human intervention.

4 A Distributed Architecture

The *OASIS* service shown in Fig. 1 implies a monolithic service including role entry, access control and the service being secured. We mentioned earlier that this need not be the case and there are strong arguments for separating an *OASIS* aware service (access control, access policy and service) from an *OASIS* certificate issuing and validation service (role entry, role entry policy, credential record storage and certificate validation).

1. The functions of the certificate issuing and validation service are common and can be shared amongst many *OASIS* services.
2. These functions are the foundation of the system's security and should be resident in a physically secured environment.
3. Validation of a certificate requires authenticated communication with other *OASIS* services. By sharing validation services the number of servers is drastically reduced and much of the communication becomes localised within a server.
4. Similarly, the reduced number of servers reduces administration problems.

The deployment of *OASIS* within an enterprise also requires consideration of availability and performance. If the *OASIS* issuing service is not available the entire distributed environment will come to a halt. The approach to take is a question of scale. In a smaller organisation where a single server can provide good

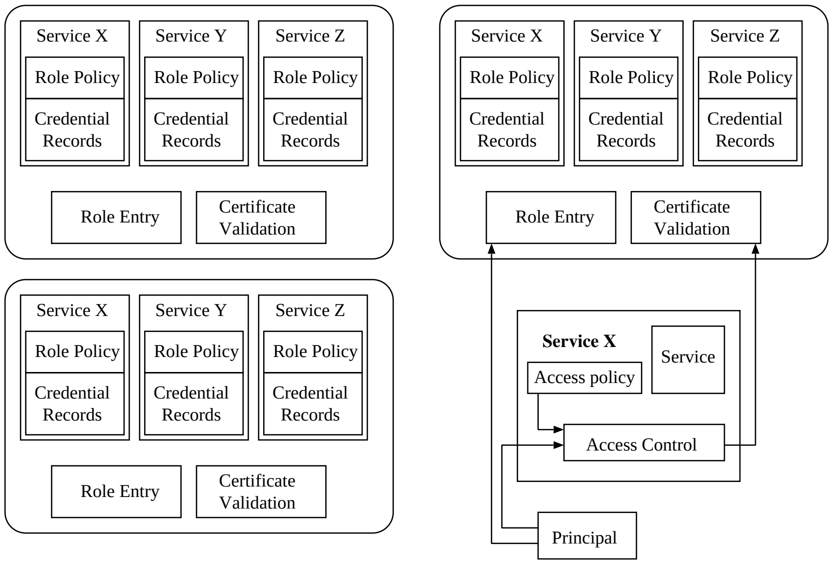

Fig. 5. An architecture for distributed *OASIS* servers.

performance a hot back up can be used to achieve availability. This technique is common and well understood. In a larger enterprise a single server may not meet performance requirements and/or the complexities of the network may invalidate the use of backup for high availability. The architecture presented in this section is designed for a single enterprise that requires a more robust and complex solution.

Our design allows the *OASIS* certificate issuing and validation servers (here after simply called servers) to be arbitrarily replicated. The algorithms below assume the credential database for each *OASIS* service is replicated on all of the *OASIS* servers. Figure 5 shows three servers with the replicated databases of three services. The service X which uses *OASIS* role based access control may use any server to validate certificates. Clients use the role entry function of the servers to obtain new certificates.

By replicating a service across several machines high availability can be achieved, and by providing multiple points of access we decrease the latency through increased concurrency and the possibility of local access. To further ensure good concurrency, our design does not depend on locking protocols to maintain consistency of the distributed data structure.

These federated servers are identical from the users' viewpoint. That is, the same result is obtained regardless of the server to which a request is addressed. We achieve this using a weak consistency model which tolerates limited transient inconsistency amongst distributed data structures at any instant in time.

Replication of the credential record graph leads to several problems that must be addressed by any solution.

Concurrent Update. Requests to update the CR graph for a particular service may occur independently and concurrently on different servers. For example, two clients may request entry to roles at the same time.

Propagation Delay. Race conditions may arise involving the propagation of a change to a CR graph and the use of that change by a client.

Partial Failure. In many systems the goal of replication is to increase availability and reduce latency. Algorithms that address partial failure are intended to restore an inoperable or disconnected server to operation. *OASIS* servers face the same problem but must also consider the implications of partial failure on access control decisions.

The following assumes that a reliable transport protocol is available between pairs of servers. Each server is assumed to have a persistent store capable of withstanding machine crashes. We assume that a technique such as a redo log allows updates to the credential database to be done atomically [7].

4.1 Distributing Updates

Each server maintains a full and complete replica of each service's credential record graph. This enables each server to validate any certificate issued by any *OASIS* service. The next concern is the maintenance of consistency across the servers.

Our consistency protocol is based on an update-notify model. Each server may make modifications to its own credential database at any time. The server then reliably broadcasts an update message communicating the change to the other servers in the federation. An update message will indicate either an addition or a removal of a CR. A functional definition of these messages is:

```
add ((CID, CRR), new CR, parent list [(CID, CRR) list])
revoke ((CID, CRR))
```

An `add` message carries a new CR, its CID and CRR, and a list of CID and CRR pairs indicating the certificates upon which this CR depends. The server is required to insert the new CR and a new pointer from each CR on which it depends. No existing information is modified.

A `revoke` message carries the CID and CRR pair of the CR which is to be removed from the tree. This CR and any CRs that depend on it are removed.

Each server maintains a sequential count of the updates it initiates. This count is included in each update message broadcast to other servers. Each server maintains a list of the update messages it has seen from all other servers. The sequence count is also used in creating the CID whenever a new certificate is created. As a consequence we have a partial ordering of all messages and CIDs.

Upon receiving an update message, a server is expected to update the appropriate graph to maintain consistency with its peers. Generally speaking the `add` and `revoke` operations are idempotent and may be independently executed in any order. However, there are two situations that can lead to one or more of the CID/CRR pairs in an update message referring to a non-existent CR in the current server's graph 1) the referenced CR has been concurrently revoked, or 2) the pair refers to a new CR which has not yet been notified to this server. In the prior case, the server should simply discard the update message since the earlier revocation implies that the current tree is already the updated version. In the latter case, the server should retain the update message until it receives the update message notifying the creation of the referenced CR.

These cases can be distinguished by comparing the sequence number in the CID with the messages seen from the originating server. If the CID has a higher sequence number it refers to a new certificate for which no CR yet exists at this server. If the CID is smaller it implies that the CR has been deleted.

4.2 Support for Fault Tolerance

In general replication is used to increase availability by allowing service to continue despite a failure. In replicating a security system great care must be taken that operation during a partial failure does not jeopardise system security. The protocol described below is designed to deal with server crashes and network partitions of both a transient and persistent nature.

We assume a fail-silent system in which a crashed server simply stops sending messages, rather than generates erroneous messages. Our aim is to provide support for fault tolerance in the protocol level in order to maintain consistency across replicated servers.

Failure Cases. The protocol described in Sect. 4.1 requires update information to be broadcast to all servers to enable them to update their credential database to reflect the current state of the service. Inconsistency is introduced if a server does not receive one or more messages. The effect of this inconsistency is possible ambiguity observed by the clients. For example, a certificate may be incorrectly rejected if the validation is done by a server that has not received notification of creation of the certificate.

Network partitioning presents a particularly difficult situation for a service supporting security. If a partition separates a set of servers into two groups, while all servers are functional in their own right, updates made to servers on one side of the partition will not be reflected in servers on the other side. The servers are active but have an inconsistent view of the state of the credential records. This has two implications:

1. The creation or invalidation of certificates on one side will not be noticed on the other side until the partition is repaired.
2. A role entered on the basis of an auxiliary credential may not be able to be revoked if the allocator resides on one side of the partition while the holder is on the other.

We have designed protocols to address each issue. A heartbeat is used to allow lazy consistency, but also detect failures and partitions. We recognise that many failures are transient and provide reliable messaging to overcome these. For more persistent failures that result in a server's database becoming significantly out of date we restore the full database from an agreed checkpoint.

Finally, we address the question of how an operating server should respond to a partial failure by deferring to policy. Some services may be prepared to operate in a partitioned state. Other services may judge that this is too great a risk. The correct place to make this decision is in the policy of each individual service. We accomplish this by providing a simple variable representing the current operational state of each server within the system. The variable may be used in expressing policy allowing decisions to include constraints such as "all servers operating" or "a majority of servers operating". Observe that while it is quite feasible to make decisions on the validity of a certificate it would be unwise to allow copies of the database on both sides of the partition to be updated. Our recovery protocol for persistent failures will not merge two separate databases.

Recovery Protocols. We use separate protocols for recovering from transient and permanent failures. Transient failures are addressed by introducing reliable message logging [1] to the messaging protocol of Sect. 4.1. Where a long term failure occurs the inconsistency of the database may be such that it is more appropriate to download the entire database from another server or to introduce a new replacement server. At this point we switch to a recovery protocol that is a form of active replication [12].

We define five states which describe the state of a server at any given instant of time.

Normal The server is operating normally. It believes all its peer servers are also alive and share a weakly consistent state.

Replay Logging A server enters this state when it has detected a failed server amongst its peers. It maintains a redo log of all messages which have not been delivered to the failed servers for replay in the future.

Down A server enters this state if it is crashed.

Recovering A server enters this state when it has rebooted following a crash, or network communication to its peers is restored after network partitioning.

Coordinating A recovering server nominates a server to coordinate its recovery phase. The nominated server enters this state.

Short term failures that cause update messages not to be delivered are handled using reliable message logging. In handling a client request to add or revoke a credential record, the server delays the reply until the update is fully logged in its persistent store. It then broadcasts the updates to the other servers. It retains a persistent copy of each broadcast message until it receives an acknowledgement from all other servers.

A heartbeat protocol amongst the servers is used to maintain the currency of connections. Where data is transmitted the heartbeat is piggybacked to avoid unnecessary traffic. When a server detects that another server is down it enters the Replay Logging state, maintaining a structured log of messages not acknowledged by the failed or disconnected server. In this state all update messages are saved until the failed server rejoins the federation or a decision is made that any future recovery will be made using a full restore.

If and when the failed server restarts it enters the Recovering state. In this state it uses the last sequence numbers seen from each server to request that all subsequent messages be resent. Once these messages have been received and acknowledged all servers can return to the Normal state of operation.

If a server remains unavailable for a sufficient period of time recovery from structured logs may be abandoned. If this happens all messages that are queued for this server only may be deleted. If this empties the queue of logged messages the server may return to a Normal state.

Where a long term failure occurs the inconsistency of the database may be such that it is more appropriate to download the entire database from another server. In this case other servers discard their redo logs and assume that if the server returns it will recover by doing a full transfer. This allows the same algorithm to support the introduction of an entirely new server and a restart of the failed server.

The full restoration of a server consists of three operations: agreement on a state, $\mathcal{S}$, to be transferred, transfer of $\mathcal{S}$, and transmission of messages arising after the establishment of $\mathcal{S}$. The new server selects one of the operating servers as the coordinator of the recovery process.

Figure 6 demonstrates this recovery protocol. A, B and C are replicated servers, and the state of each server is denoted by S_A, S_B and S_C. Initially, S_A, S_B and S_C are identically equal to $\mathcal{S}$. Let P be a principal requesting a certificate from A. A validates the request and creates a certificate c for P. It will then send an update message $upd(c)$ to both B and C. Now, suppose C crashes before receiving $upd(c)$. Detecting the loss of contact with C, both A and B will move from Normal state to Replay logging state, and log subsequent messages for forwarding to C.

At this point, we know there is some failure in our federation of replicated servers. It is possible the failure is transient. If C resumes while A and B are in Replay logging state, the recovery only involves re-sending any logged messages. However if C remains down for an extended period of time, A and B will abandon this and return to the Normal state. A serious fault has been detected and C has left the federation.

When C returns it enters the Recovering state and broadcasts a message seeking a partner to coordinate its recovery. It accepts one reply, in this example B. B is responsible for coordinating all operating servers to ensure that state S_C is consistent at the end of the recovery protocol. B enters the Coordinator state.

On assuming the role of coordinator, B broadcasts a checkpoint request to all other operational servers, in this example, A. The checkpoint request indicates that C is recovering. Each server responds to the checkpoint request by providing

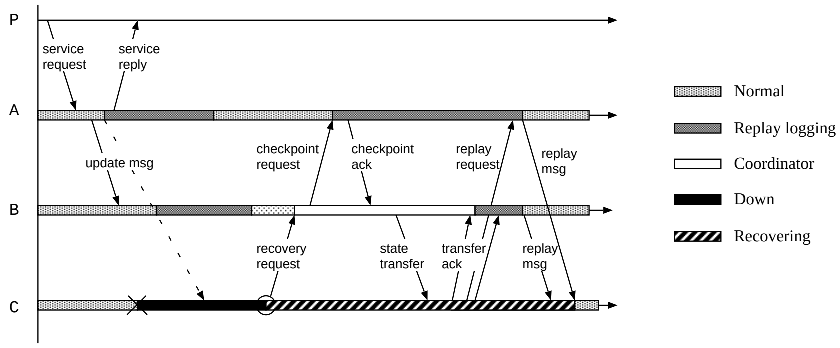

Fig. 6. Recovery from a server crash.

to the Coordinator the sequence number of the last message that it broadcast and entering the Replay Logging state on behalf of the recovering server, C.

The Coordinator awaits the responses from all servers. When these have arrived it confirms that its own database contains all the updates less than or equal to the checkpoints it has received. (The assumption of an underlying reliable messaging protocol implies that this should have happened as a matter of course.) At this point it transfers its entire database to the recovering server, C. When this is acknowledged the Coordinator has finished. It now enters the Replay Logging state, logging any further updates.

The recovering server, C, now completes the recovery process as it would recover from a transient fault. It requests all logged messages from each server specifying the sequence number reached in its database. This is just the sequence number provided by the server to the Coordinator and all subsequent updates are now provided directly to C.

4.3 Analysis

Our protocol solves the concurrent update problem by ensuring the consistency of the final state after all updates. Take the simple example shown in Fig. 7. Suppose we have two servers, A and B, replicating a CR tree. The initial state is shown in (a). Assume a request is made to A to revoke CR4, and simultaneously another request is made to B to create CR6 dependent upon CR4. Both A and B would proceed with their individual requests, producing a temporarily inconsistent state, as shown in (b).

However, A and B will send each other an update message after the local modification has been made. A will notice that CR4 is no longer valid because it was revoked, and it will discard the update message. A will also increment its view of B's current sequence number, since B has used the sequence number to create CR6. B will remove CR4 as a result of receiving the update message from A, producing a tree which is identical to A's. This is the final state of the consolidated CR tree, as shown in (c).

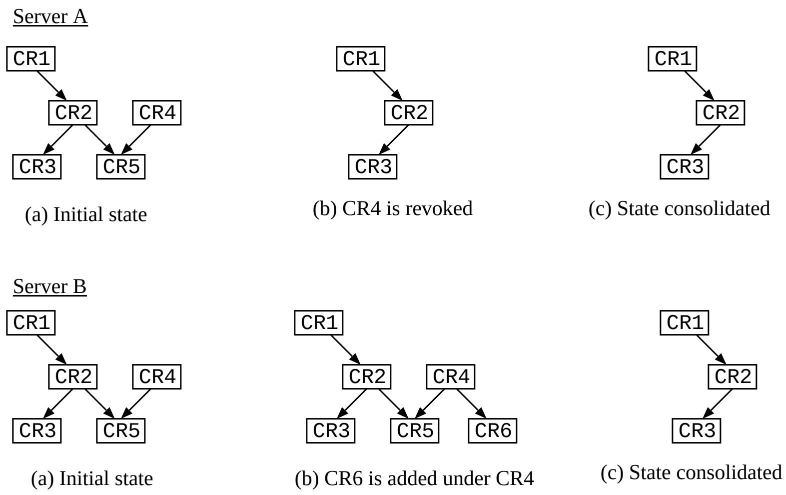

Fig. 7. Resolution for concurrent updates.

One may argue that there is a window of chance that the certificate for which CR6 is a credential will be used before the revocation of CR4 takes place. If the certificate is presented to A, it would treat it as a new unknown certificate. After the reception of the update message from B, it will learn that the certificate depends on a revoked CR, therefore the use of this certificate will be rejected. If this certificate is presented to B, B will authorise its use if the update message from A has not been received. We argue that this problem is an example of a race condition. Even in a single server situation, the order of arrival of the requests determines the exact consequences. There would still be a chance that certificate CR6 may be used before it is revoked.

Our protocol also solves the problem of delayed propagation. The servers' sequence numbers allow each server to determine whether a missing CR has been revoked or is new. The servers can take appropriate action. For example, a certificate issued by A may be sent for validation before B receives the update message. If B has been asked to validate this certificate, it can determine that this is a new CR. In this case it would wait for the update message from A.

Correctness for Recovery Protocol. Our recovery protocol is correct because of three crucial attributes: the update operations are commutative; every update is known by at least one server within the federation at any given time; and our assumption of a weak consistency model.

Updates made to a CR tree are either an addition of a new CR or a removal of an existing one, with removal collapsing all dependent CRs. It is trivial to see the commutativity if two updates are made on unrelated CRs. If two updates are made on related CRs, this property still holds. Consider adding a CR that depends on a CR which is being revoked. The addition will fail since there is no CR for the new one to depend on in the tree. If the two operations are done in reverse order, the final result would still be the same. This is exactly the case illustrated in Fig. 7.

Consider the case where two additions are made concurrently to a server, with one creating a CR that depends on the CR created by another. If the dependent CR has not been processed by the server, the addition will be left pending until it has been created. This in effect guarantees a consistent final state by serialising the addition operations. The last case involves two removal operations made on related CRs. Removing a node of an inverted tree and its dependents and subsequently removing a second node that the first depended on results in exactly the same tree as removing just the second node.

The commutative property of update operations is crucial since our recovery protocol is based on replaying messages and this works regardless of the order of replay.

The requirement that a server must fully log an update before replying to the client ensures that no orphan state can exist [1] and [5]. Therefore recovery is a straightforward transfer of state followed by a resend of update messages. This scheme also works in multi-node failure situations.

5 Conclusion

We have presented a description of *OASIS* role based access control and a distributed architecture for supporting it. Work continues on both the functionality of *OASIS* and on the architecture. Our understanding of auxiliary credentials and the different ways in which they may be applied is still evolving. This will improve with experience applying role based access control in different contexts.

The protocols employed in the architecture presented here retain a relatively high level of efficiency at the cost of replicating all services on each server. This constrains the scale that can be achieved in several ways. The cost of updates is proportional to the square of the number of servers. Partial failures may cause some services to stop functioning when they could have functioned quite happily with a smaller number of replicas all on operating servers. A next goal is to produce a design that allows a subset of services to be replicated at each server.

An implementation of the distributed model is under way. It will be used to address questions about the cost of protocols and the system's ability to scale. It will also be used to gain experience with the organisation of roles. We are also investigating the use of *OASIS* role based access control in areas of electronic health records and network management.

Role based access control promises a number of advantages for security in large, complex distributed systems. In this paper we have presented a distributed architecture that supports a resilient, highly available access control service based on the *OASIS* model. This includes protocols supporting weak consistency and recovery from server and network failures.

References

1. L. Alvisi, B. Hoppe, and K. Marzullo. Non-blocking and orphan-free message logging protocols. *23rd Int. Conf. on Fault-Tolerant Computing (FTCS-23)*, pages 145–154, 1993.
2. David F. Ferraiolo, John F. Barkley, and D. Richard Kuhn. A role-based access control model and reference implementation within a corporate intranet. *ACM Transactions on Information and System Security*, 2(1):34–64, Feb 1999.
3. Richard Hayton. *OASIS An Open Architecture for Secure Interworking Services.* PhD thesis, Fitzwilliam College, University of Cambridge, Mar 1995.
4. Richard Hayton, Jean Bacon, and Ken Moody. Oasis: Access control in an open, distributed environment. In *Proc. of IEEE Symposium on Security and Privacy*, pages 3–14, Oakland, CA, May 1998. IEEE.
5. D. B. Johnson and W. Zwaenepoel. Sender-based message logging. *17th Int. Symp. on Fault-Tolerant Computing*, pages 14–19, 1987.
6. B. W. Lampson. Protection. In *Proc. Fifth Princeton Symposium on Information Sciences and Systems*, pages 437–443, March 1971. reprinted in Operating Systems Review, 8, 1 (Jan. 1974) pp. 417-429.
7. Tobin J. Lehman and Michael J. Carey. A recovery algorithm for a high-performance memory-resident database system. In *Proceedings of ACM SIGMOD Annual Conference on Management of Data*, San Francisco, May 1987. ACM.
8. R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.
9. Matunda Nyanchama and Sylvia Osborn. The role graph model and conflict of interest. *ACM Transactions on Information and System Security*, 2(1):3–33, Feb 1999.
10. J. H. Saltzer. Naming and binding of objects. In R. Bayer, R.M. Graham, and G. Seegmuller, editors, *Operating Systems, An Advanced Course*, pages 99–208. Springer-Verlag, Berlin, 1979.
11. Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38-47, February 1996.
12. F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.
13. J.G. Steiner, C. Neuman, and J.I. Schiller. Kerberos: An authentication service for open network systems. In *USENIX*, Dallas TX, 1988. Uniforum.
14. Gene Tsudik. Message authentication with one-way hash functions. In *IEEE Infocom 1992*. IEEE Press, May 1992.