Cardea:  
An Approach to Distributed Authorization

A Master’s Paper in  
Computer Science  
By  
Rebekah Lepro Metz

©Rebekah Lepro Metz

Submitted in Partial Fulfillment  
Of the Requirements  
For the Degree of  

Master’s of Science

December 2004
Abstract

The emergence of *Service Oriented Architectures (SOAs) and Virtual Organizations (VOs)* is changing the basic interaction and control patterns in computing infrastructures. Adapting to this new paradigm requires an authorization model that considers interactions that span organizational boundaries, relies on different authentication sources, and manages complex information as part of the authorization process. This thesis presents an authorization framework based on such an evolutionary model. *Cardea* enables dynamic access control for distributed systems through a service oriented approach. Cardea adopts open Internet standards such as SAML, XACML, WSS, and XML Digital Signature to establish an authorization process that supports access control across distributed resources. Cardea distills the requirements of a distributed environment into several architectural tenets that underlay multiple aspects of the framework. Adherence to these tenets encourages modularity, portability, interoperability, scalability, and reuse of existing technology infrastructure within the authorization framework itself. Ultimately, Cardea enables a flexible and scalable authorization infrastructure that addresses the unique requirements of distributed systems security.
# Table of Contents

Abstract................................................................................................................................i

Table of Contents ............................................................................................................... ii

List of Figures ................................................................................................................... iv

Acknowledgement ...............................................................................................................v

1 Authorization in a Distributed Environment ............................................................ 1

  1.1 Process Model.......................................................................................................... 3

  1.2 Evolution of the Authorization Process Model .................................................... 5

  1.3 Exemplifying the Evolution of the Authorization Scenario ................................ 5

  1.4 Requirements for a Distributed Authorization Process Model....................... 11

  1.5 Architectural Tenets for a Distributed Authorization Process Model.......... 13

  1.6 Relevant Standards and Specifications............................................................... 16

    1.6.1 SOAP ........................................................................................................ 17

    1.6.2 XMLDSig ................................................................................................. 17

    1.6.3 WSS .......................................................................................................... 18

    1.6.4 SAML ....................................................................................................... 19

    1.6.5 XACML .................................................................................................... 20

  1.7 State of the Art ...................................................................................................... 21

    1.7.1 SOAP Messaging...................................................................................... 21

    1.7.2 XML Message Security ............................................................................ 22

    1.7.3 Security Information Representation ........................................................ 24

    1.7.4 Identity Management Frameworks ........................................................... 28

    1.7.5 Security Toolkits....................................................................................... 32

    1.7.6 Alternate Authorization Approaches ........................................................ 36

2 Cardea – A Departure from Traditional Approaches............................................. 40

  2.1 Key Design Issues of Cardea................................................................................ 41

    2.1.1 Communication paradigm......................................................................... 41

    2.1.2 Information representation........................................................................ 41

    2.1.3 Implemented authorization algorithm....................................................... 42

    2.1.4 Authority binding........................................................................................ 42

  2.2 Initial Prototype Features .................................................................................... 42

  2.3 Architecture........................................................................................................... 43

    2.3.1 Platform implementation ......................................................................... 47

    2.3.2 Web services ............................................................................................. 48

    2.3.3 Standards conformance............................................................................. 48

    2.3.4 Modes of operation ................................................................................... 49

  2.4 Pre-requisites and Assumptions ........................................................................... 50

  2.5 Information Representation.................................................................................... 51
List of Figures

Figure 1. Conceptual Phases of Authorization ................................................................. 4
Figure 2. Individually assigned permissions ................................................................. 6
Figure 3. Group managed permissions ........................................................................ 7
Figure 4. Multiple services in a single security domain ............................................ 8
Figure 5. Multiple services in a multiple security domains ....................................... 9
Figure 6. Cross-domain configuration synchronization ........................................... 10
Figure 7. WSS block and critical relationships ......................................................... 19
Figure 8. The XACML processing model (simplified) ................................................. 20
Figure 9. Cardea Architecture .................................................................................... 44
Figure 10. Internal PDP Architecture ......................................................................... 45
Figure 11. Cardea Message Structure ......................................................................... 46
Figure 12. Cardea Software Components ................................................................. 47
Figure 13. Service Agreements to Establish Trust .................................................... 57
Figure 14. Communicating with the appropriate PDP ............................................. 61
Figure 15. PEP communicates directly with PDP ..................................................... 63
Figure 16. PDP receives request from principal and forwards decision to PEP ....... 64
Figure 17. Principal requests authorization PDP and forwards credential to PEP .... 65
Figure 18. Signature generation and verification model ........................................... 67
Figure 19. A sample SAML Exchange .................................................................... 93
I would like to express my gratitude to Dr. Linda Null for her valuable insight and encouragement along the way. I would like to thank NASA and Computer Sciences Corporation for their financial support. I would like to thank Derek Simmel for his insightful comments and thoughtful review. Finally, I would also like to thank my husband without whom this work would never be.
1 Authorization in a Distributed Environment

Authorization is the process by which an entity is granted permission to access specific resources. The goal of the authorization process is to enforce the security policies of the enterprise that controls a resource by correctly applying those policies to authorization requests. Current authorization processes were originally designed to manage permissions in an environment protected at the edges and centrally controlled within that perimeter. Within this boundary, a small number of authorities evaluate requests from a limited set of users to access a known set of resources. With the advent of service-oriented architectures (SOA) and grid computing, a “System of Systems” view of computing infrastructures is emerging. Now, an entity from one enterprise may request access to resources controlled by an external autonomous enterprise. Subsequently, authorization processes must control access across dynamic Virtual Organizations (VOs), which often span organization and network boundaries. Therefore, the environment within which authorizations are managed has evolved into a pervasive, highly dynamic context.

Many assumptions made about this context no longer hold. As tasks and problems tackled by computing systems evolve in complexity, support infrastructure for those tasks must also evolve. They must take advantage of heterogeneous and geographically distributed resources. They must utilize the processing power of many individual systems to achieve a single common goal. They must share information globally. They require coordinated access to resources on-demand [20] Thus, security processes, including authorizing processing, must now address the requirements particular to these needs. Conceptually these requirements include:
• Providing a robust authorization decision process that spans multiple independent security domains [13]

• Allowing participating authorities to negotiate an authorization decision online without restricting functionality or policy internal to a particular security domain

• Ensuring that all participants in a single authorization transaction can trust and interpret information that crosses administrative boundaries

• Reducing reliance on specific identity credentialing and enforcement mechanisms.

This thesis presents an authorization approach that addresses these unique requirements. This approach provides a distributed yet cohesive means to coordinate authorizing transactions within a VO, and thus may span organizational and network boundaries. Therefore, the approach is inherently scalable, as it does not rely on perimeter or network boundaries to partition its problem space. This approach also leverages a robust attribute-based access control policy model that can natively account for the dynamic and complex context of access requests.

The remainder of this section explores the problem of authorization in a distributed environment. An extended example illustrates the evolution of the problem space in Section 1.3. Then several of the issues underlying the distributed authorization problem space are examined in sections 1.4 and 1.5. Section 1.6 introduces the specifications incorporated into Cardea’s processing model and highlights important characteristics of their scope, design, processing model and syntax. Section 1.7 reviews the current state of the art for distributed computing, enterprise security, and distributed authorization systems. The second section presents the Cardea system. Section 2.1
briefly outlines the key design issues that shaped the system. Section 2.2 describes the features of the initial prototypes. Section 2.3 explains Cardea’s architecture. Sections 2.4 through 2.9 present Cardea’s design and processing model in detail. The final sections, 3 and 4, recap the benefits Cardea provides to the distributed authorization domain and identify potential directions for future research that would augment the research presented here.

1.1 Process Model

Authorizing is the process by which an entity is granted permission to access specific resources. It ensures that every access is controlled and that only authorized access takes place [69]. Authorization can be modeled in a variety of ways. As we examine the unique requirements on authorizing within virtual organizations, we model the phases of the authorizing process. The authorization process is modeled as a request-response transaction between a requesting entity and an enforcement mechanism that can be partitioned into four distinct phases. These phases are:

1. Initiating the request
2. Evaluating the request
3. Reaching a decision
4. Enforcing that decision

Figure 1 illustrates how the steps of an authorization process fit into these phases. When governance of the process lies with a VO, responsibility for each phase can be distributed to autonomous organizations within the same VO. Therefore, orchestrating communication and context between these independent players becomes a core function of the process.
Traditionally, authorization processes are self-contained within a system or within an organization. As such, they tightly couple policy evaluation and enforcement mechanisms, effectively combining the second, third, and final phases of an authorization transaction. The processes offload many security tasks to perimeter-based security mechanisms such as firewalls. Furthermore, they presume a controlled environment in which they can fully predict the potential interactions between a known set of users and finite number of resources. This is not to say that current authorization approaches cannot handle any change. However, adapting to changes in the environment often requires modifications that address specific changes. Therefore, each workaround increases the overall complexity of the operational environment by creating additional
specific conditions that must be individually addressed and maintained.

### 1.2 Evolution of the Authorization Process Model

To bypass these limitations, a new approach to offering the authorization process in a distributed environment must emerge. Cardea\(^1\) addresses the deficiencies of existing authorization approaches when applied to a VO environment. First, Cardea adopts an attribute based access control policy model. Then, Cardea models authorization independently from supporting authentication infrastructure. Thus, it builds upon existing identity management infrastructure investments. Further, Cardea’s model decouples the decision and enforcement phases of an authorizing transaction. Therefore, authorization decisions can be negotiated at the VO level and enforced by independent mechanisms within the boundaries of each VO participant. This approach facilitates a global yet decentralized view of authorization that does not mandate local functionality and practices. Finally, Cardea standardizes the methods to exchange trusted information across organizational boundaries within a virtual organization.

### 1.3 Exemplifying the Evolution of the Authorization Scenario

The following extended example of requesting permission to access a file from an FTP service illustrates some of the authorization problems that arise during the evolution to a SOA or VO environment. Initially, authorizations are assigned to individual entities to access individual resources. For example, a simple FTP service, as depicted in Figure 2. Individually assigned permissions, can grant individually named users the permission

---

\(^1\) In Roman mythology, Cardea was the goddess of thresholds and door hinges. Her name comes from Cardo meaning door-pivot. She was worshipped at the Beltane festival and during June, both of which were seen as a metaphorical “hinge” for the year. Ovid said of Cardea, “Her power is to open what is shut; to shut what is open.” Source: Wikipedia [http://en.wikipedia.org/wiki/Cardea]
to access specific files.

Figure 2. Individually assigned permissions
As more users want to access the files available from the FTP service, the burden to manage an expanding user base increases significantly. To reduce this burden, users are often partitioned into groups according to their common needs or according to the roles\textsuperscript{2} they play within the organization [29]. Permissions appropriate to those needs can then be assigned indirectly to users via group memberships or directly according to identity.

\textsuperscript{2} Assigned roles are often represented as group to which a user belongs
Figure 3. Group managed permissions

Figure 3 depicts an FTP service that manages privileges according to identity and group. For example, user U1 accesses a file A via an individually assigned permission whereas user U4 accesses the same file via permissions assigned to group G1. Similarly to partitioning users into groups, resources can also be partitioned to divide the management problem. In this example, Penn State’s FTP service has limited disk space. Therefore, Penn State stands up a second FTP service and partitions the files between those services. Even if a common front end hides the partition from the end user, each service must be separately configured with permissions appropriate to each member of the Penn State community, as illustrated in Figure 4.
Users may also want to access resources beyond those available within a single organization. For example, in Figure 5, a user at Penn State wants to access a file published by NASA. This file is available from NASA’s FTP service. Rather than waiting for a copy of the file to be added to one of the Penn State FTP services, the user wants to access the file directly from NASA. To deliver the file to a Penn State community member, the NASA FTP service must create a local identity representation for that user that is not correlated to the original user identity within Penn State’s organization. For example, User1 within NASA and U1 within Penn State both represent the same entity.
For NASA and Penn State to join together as a single VO, their configuration states must be synchronized. For example, if a group of researchers at NASA and Penn State wants to collaborate on a project and share project files via the FTP services, each site must internally understand the users from each domain that are collaborators. Each site therefore needs to independently create identities for users in both organizations and assign permissions to those identities. Even if groups are used; identities and permissions, and groups must still be synchronized across organizations. In the example, a group \(G1\) is created at Penn State to represent project collaborators and a group \(\text{group1}\), is created at NASA to represent those same project members. If project member \(U4\) at Penn State (User 4 at NASA) resigns from the project, Penn State must update the definition of \(G1\). NASA must also update the definition for \(\text{group1}\). Otherwise, \(U4\) could still access the project resources hosted at NASA.
As resources become potentially available to all members of a VO, the number of potential authorization requests that must be managed within each organization increases dramatically with the advent of a VO if organizational boundaries initially partitioned users and resources into small groups. Two alternative approaches to managing this explosion of potential resource request types have emerged. The first approach synchronizes authorization mechanisms across organizations in some out-of-band manner. This approach introduces a heavy configuration burden and therefore a limited ability to scale. The second approach considers the VO itself as the authoritative organization that manages identities rather than independent organizations. This approach usurps self-determination from the independent domains, promotes conformity over interoperability, and ultimately remains inflexible. In contrast, a native distributed authorization model addresses such limitations on scalability, flexibility, autonomy, and interoperability of existing approaches.
1.4 Requirements for a Distributed Authorization Process Model

An analysis of this evolution identifies several characteristics that differentiate authorization in distributed system from the more traditional authorization scenarios.

- Authorization in a distributed system requires widespread cooperation across organizational boundaries.
- The authorization process participants must negotiate an authorization decision in-band [70].
- Cooperation and negotiation must not restrict the functionality or policy within organizational boundaries [13].
- Each participant must be able to verify and interpret information exchanged within the VO [43].
- Decisions must not be tightly coupled with infrastructure mechanisms. [36]

In support of cooperation and negotiation within a VO, each participating organization must be knowledgeable of shared VO state. Rather than duplicating data within each organization, the model should include mechanisms to discover the majority of necessary information within the context of an authorization transaction. For example, each organization should establish out-of-band which VO authorities it trusts to verify identity rather than directly determining each trustworthy identity directly. Then, an organization can vet a particular identity by communicating with those authorities. Minimizing mandatory yet duplicate state representation translates directly into increased scalability. To allow each VO member to utilize global state information as internally appropriate, the selected information representation must not presume specific internal
mechanisms.

It follows that a VO member must be able to easily interpret and verify any discovered information. Thus, exchanged data must carry established and widely accepted syntax and semantics. Otherwise, consuming the information requires each member to negotiate meaning individually with every other member to ensure correct understanding—doable, but impractical and not scalable. Within a decentralized approach, new members emerge and obsolete members withdraw as necessary. Continued members must not be required to negotiate unique information exchanges with each new member nor be required to decommission support for legacy modes. At a minimum, distributing an authorization transaction requires common communication protocols, syntax, and decision semantics.

An authorization process relies on authentication and enforcement mechanisms to support the decision process. To support their internal authorization process, organizations will have typically invested in identity and enforcement infrastructures particular to their security needs before joining a VO. There are no guarantees that members of a single VO will share similar infrastructures. Therefore, the authorization process cannot presume a particular type of mechanism [13]. In fact, a distributed authorization process must also support complex access decisions that involve a variety of resources from different administrative boundaries without extensive modification to or replacement of existing infrastructures. Further, enforcement within a single domain must not rely on similarities to or mechanisms in any external domain. Finally, the VO
An authorization process for distributed computing solutions that share resources across domains must address requirements particular to this shared environment [13]. First, each domain must remain as autonomous as possible. Further, the approach should not impose a physical nor conceptual point of control on collaborating domains. For example, a principal’s access from Pittsburgh to resources in Chicago should not be held hostage by availability of authorization data in San Francisco. The resources in Chicago should have or be able to accumulate enough data available locally to authorize access requests from Pittsburgh. Moreover, the process must maximize the amount of data discoverable within the context of each independent transaction, thus minimizing the amount of locally managed duplicate data. Finally, the approach must remain independent of infrastructure that implements the approach in each domain.

### 1.5 Architectural Tenets for a Distributed Authorization Process Model

Satisfying the requirements of authorization in a distributed environment must address the problems faced by any distributed system, the problems faced by any authorization system, and the problems specific to a distributed authorization system. For example, determining authoritative sources of information, establishing messaging protocols, formatting messages, and providing security are tasks common to any distributed system. Furthermore, managing authorization information, making the correct authorization decision, and enforcing that decision are critical features of any authorization system. Finally, representing a single decision that can be enforced by
diverse enforcement mechanisms is particular to the distributed authorization problem space.

There are several critical characteristics that are common to these issues. Although these characteristics may manifest differently at different levels, the selected solution can be re-applied to build solutions. These issues include:

- Standard language (syntax, grammar, and unambiguous definitions of terms and operators), for shared comprehension
- Standard protocol (rules of conversation), for consistent, reliable interpretation
- Modularity & interoperability, for diversity of implementation
- Portability & platform independence, for flexibility and broader usability

Establishing a common language for data exchange enables common semantics across diverse participants in a VO. Certain entity attributes, such as organizational role, must be expressed in a manner understandable to all VO participants while other attributes, such as salary might remain proprietary to a single organization. A common format allows data to remain independent from the internal processes that generate and consume it. This separation increases autonomy in selecting appropriate internal infrastructure and reduces the configuration requirements stemming from data interpretation. Further, it encourages shared comprehension despite heterogeneous implementation.

After establishing a common language for critical data, any distributed system
must define a protocol to that share data in that language among components. This protocol must be widely deployable across heterogeneous environments; be extensible to handle new and unforeseen collaboration scenarios; and facilitate discovery of information directly within the context of a transaction. These requirements support a variety of underlying messaging infrastructure implementations while ensuring consistent and reliable interpretation of shared data. For example, a web browser on a Windows platform can consistently communicate with a web server hosted on a UNIX server to exchange html data.

Modularity and interoperability are central tenets of distributed systems. These design paradigms encourage flexibility and scalability via diversity of implementation. However, in existing authorization systems, several functions are often tightly integrated within a single component. This limits the ability to distribute responsibility for specific functionality across a VO. For example, mechanisms often bind, via pre-configuration of user account attributes, access control lists or mapping files, a local identity representation with a definition of access permitted to local. Such binding does not scale in a distributed environment in which transactions involve a continually evolving user and resource base. For example, grid-computing scenarios pertain to potentially tens of thousands of users that will actively use resources available across multiple security domains participating in a single grid [23].

Architecting for dynamic computing scenarios that span organizational and network boundaries requires consideration of flexibility and broader usability. Relying on
mechanisms bound to local infrastructure limits component portability and platform independence. For example, any solution that requires X.509 identity certificates cannot be ported to an organization that does not have a PKI infrastructure.

### 1.6 Relevant Standards and Specifications

Three predominant factors motivate compliance with open standards in a distributed authorization system:

1. The system can build upon the research and debate that goes into any standardized solution to a general problem. Therefore, the adopted solution inherently considers the issues of security, flexibility, reliability, and scalability.

2. A standard specifies common functional characteristics that a compliant system will display without mandating a mechanism to implement those characteristics. This ensures that the final solution allows for a variety of compliant implementation options. Therefore, a domain can select from a range of compliant solutions, dependent upon their internal requirements.

3. Adoption of the same standard encourages two domains to more easily interface in the specified manner. Therefore, standards compliance increases interoperability by abstracting the details of implementation and focusing on interactions.

Several standards currently maintained or under development by both the World Wide Web Consortium [85] and OASIS [55] focus on the approaching problems faced by a distributed authorization system. The bodies maintain authoritative sources of information on those standards and their nuances. However, below is a brief overview of
several of the relevant standards and how they each fit into the technology stack for a distributed authorization system. These standards include the SOAP Protocol [8], the XML Digital Signature Recommendation (XMLDSig) [3], Web Services Security (WSS) [52], the Security Assertion Markup Language (SAML) [24], and the Extensible Access Control Markup Language (XACML) [22]. Each of these standards defines a common language in XML [9] format for representing security data pertinent to authorization, and outlines functionality to manipulate that data. These standardization efforts focus primarily on exchange formats and patterns for secure messaging within a SOA. As such, they are necessary but not sufficient components to providing an end-to-end authorization infrastructure.

1.6.1 SOAP

SOAP defines a standard protocol that layers on top of existing transport protocols to exchange messages within a SOA. For example, a SOAP message may be bound to the http protocol. A SOAP engine generates SOAP-compliant messages, forwards them to its intended recipients, and manages SOAP-specific processing and communication errors. Each SOAP message conforms to a generic envelope format. This generic envelope format provides a basis to exchange arbitrary and complex messages between collaborators independently of their internal infrastructures. Therefore, a SOAP engine can process any SOAP message regardless of the data carried in the message.

1.6.2 XMLDSig

The XML Digital Signature Recommendation specifies syntax and processing rules for providing data integrity, messaging authentication, and/or signer authentication
in an XML context. XMLDSig also specifies how to encode the resultant signature
details into a native XML structure. In addition to specifying how to generate and
represent information pertaining to signed XML content, XMLDSig identifies three
standard methods to associate a signature with the signed content. Finally, XMLDSig
provides the ability to sign specific portions of an XML construct. Therefore, different
entities can sign distinct portions of a single document without affecting the data
integrity. Signature detail content relative to one signature does not affect the signed
content for any subsequent signature assigned. Incorporating these security features into
the messages exchanged within a SOA provides a flexible basis of trust for the system.

1.6.3 WSS

Although XMLDSig specifies how to generate security tokens for arbitrary XML
content, it does not standardize how to associate those tokens with a SOAP message.
WSS syntax defines an encoding to include SOAP message security and signature
information directly within the SOAP message headers. WSS also standardizes SOAP
body signatures. Finally, the standard addresses how to bind various authentication
tokens, such as a userid/password pair or an X.509 digital certificate, and to locate those
authentication tokens for verifying the message signature.
The WSS syntax uses the XMLDSig syntax to represent the signature information within a message. Figure 7 illustrates the relationships between the XML constructs of WSS syntax and how those constructs work together to protect the payload of the SOAP message, contained in the `<SOAP:Body>` element.

### 1.6.4 SAML

The OASIS SAML standard uses XML to package and exchange security information within assertions. These assertions are the basis for exchanging guaranteed data. SAML is purposely designed to work as a background technology, defining only how assertions are communicated between entities. A single exchange may contain SAML assertions and protocol messages for exchanging attributes as well as authorization decisions. The specification places no requirements on how assertions are generated, or on the mechanisms used to verify and consume an assertion. Therefore, distinct administrative domains can maintain appropriate internal mechanisms and still effectively communicate with other domains to during an authorization transaction.

SAML assertions also provide a mechanism that permits different entities to assume
different conceptual roles within a single authorization transaction. Appendix A illustrates a SAML message exchange pattern.

1.6.5 XACML

The OASIS XACML standard provides a general-purpose mechanism for expressing and evaluating the access control policies for an organization. At its core, XACML is an access-control policy language that allows standard specification of rules about who can do what and when.

**XACML Processing**

![XACML Diagram](image)

Typically in XACML, a policy decision point (PDP) must determine the policy for a policy enforcement point (PEP) to enforce on a resource. Figure 8 contains a simplified overview of the XACML conceptual processing model. The PEP forwards the...
request to the ContextHandler\textsuperscript{3}. The ContextHandler processes the attributes provided by the PEP, retrieves attributes as necessary from one or more Policy Information Points (PIP); and forwards the request to the PDP. The PDP determines the policy applicable to the request and evaluates the rules contained in the policy to reach an authorization decision for the request. The PDP returns the decision for appropriate enforcement. By adopting this processing model, XACML conceptually separates the decision and enforcement phases of the authorization process. XACML also defines, within the policy syntax, a way to distribute responsibility for each rule to the appropriate authority and access the rule in-band.

1.7 State of the Art

The breadth and depth of requirements on a fully functional distributed access control system compels a resolution for a variety of issues that are general to distributed system architectures. These issues include selection of a communication protocol, data representation, identity management, and data security. The following subsections present a range of different approaches to provide these functions. Then, several toolkits to use as the building blocks of a distributed authorization system based on standard web services are presented. Finally, several of the alternative approaches to distributed authorization are profiled.

1.7.1 SOAP Messaging

There are several different architectural solutions adopted by SOAP engine providers currently in the marketplace. The first approach provides a SOAP API to

\footnote{The PEP and the ContextHandler are often combined into a single physical component.}
invoke directly in an application. This approach supports web-services aware applications. It provides libraries or toolkits that expose language specific APIs that supports SOAP specific functionality within the context of a familiar language or platform. The SOAP::Lite [41] toolkit and the Microsoft SOAP Toolkit [38] follow this approach. The second approach provides SOAP messaging as an extension to existing application server technologies. Established application providers, such as BEA, IBM, Oracle and Microsoft provide necessary add-ins to expose existing application logic as web services. There are also open source offerings that follow this design pattern, such as the Apache Axis Engine [88]. Each hosted service engine interacts with its host environment via hooks into the internal architecture of existing product offerings or via a standard or published API. A final approach offers a standalone SOAP engine that can accept SOAP messages and self-manage the infrastructure required to process those messages. The capabilities of such a stand-alone engine typically focus on providing specialized functionality natively exposed as a web service. For example, the Data Interchange [8] product from Cape Clear focuses on applying XML structure to existing semi-structured data. Alternatively, RogueWave Software [66] offers the LEIF framework that focuses on exposing C/C++ business logic as web services. It built the necessary infrastructure support, including adherence to the Java Servlet API, to offer those capabilities on the C/C++ platform.

1.7.2 XML Message Security

The SOAP standard does not address the concept of SOAP-level message security. Rather, it is left to implementers to select an appropriate approach for their use
model. Much like SOAP engines adopt one of several potential architectural approaches for messaging, these implementations typically also select an XML security models. Depending on the SOAP engine selected, an XML message security product may support multiple models. These models range from reliance on security of the underlying transport to integrated security based on WSS processing. The two predominant paradigms for message security in the XML layer are termed proxy and agent approaches.

A proxy approach deploys an XML firewall dedicated to SOAP security processing. This firewall intercepts SOAP messages after they pass through the transport level firewall. It consumes attached security information and forwards the message to its targeted web service. Therefore, it can apply standard security processes to messages regardless of the destination web service. For example, VordelSecure [82] can be deployed as a reverse proxy on a single server using an XML firewall deployment model. The firewall is responsible for signing, storing, and returning any response from the web service. The firewall is also responsible for returning a SOAP Fault message to the consumer if any security rules processing fails.

An alternate approach is the XML agent mode. With this approach, XML message security is deployed at the application level and its processing is distributed throughout the enterprise. With an XML agent model, a message arrives at the targeted business system using the enterprise SOAP infrastructure. However, the agent intercepts the message and processes the message security data before passing the payload to the
actual business system. For example, the WSS4J (Web Service Security for Java) Axis handler [87] specified in the deployment descriptor for a specifically deployed web service follows the agent model. Each web service application independently configures agent behavior. The agent is responsible for generating faults when processing fails.

### 1.7.3 Security Information Representation

Consistent execution of any process depends upon data input formatted in some standard representation. Security processes, such as authorization, depend upon data representing identity, characteristics, and policy. Although the decision process must know how this data will be represented within an access request, it does not define how to process and share information in an authorization decision. Rather it is left to implementations to generate, transfer, accept, and consume data packets that comply with these representations.

#### Identifiers

When developing access control mechanisms, it is typical to precede authorization with the authentication of subjects. Authentication is the process by which an entity proves a claim regarding its identity to one or more other entities (e.g., proving that you are who you claim to be). There are many methods used to authenticate subjects and various strengths to these methods. Authenticating identity is typically a function of demonstrating ‘what you know’, ‘what you have’ or ‘what you are’ as proof. With any of these approaches to authenticating identity, each individual system must establish protocols for storage, generation, and verification of the actual assigned identifiers.
Digital signatures are applied to protect information so that the authenticity of sender identity that can be recognized or verified. However, digital signatures rely on knowledge of a shared secret key used for decrypting an encrypted message. Preservation of the secret key presents a unique set of challenges, particularly when the secret must be shared with a set of participants. Public key (a.k.a. asymmetric) cryptography evolved as a way to transmit private content without also transmitting the shared secret key needed to unlock the content by relying on a complementary pair of generated keys. Conversely, a private key holder can encrypt content that can be decrypted with the public key\(^4\). Notably, one can assure that the message actually comes from the sender (that must have access to the appropriate private key) to encrypt the message content. Therefore, possession of the private half of a key pair is used as proof of identity of the holder.

A digital certificate [26] is an affidavit of authenticity regarding an entity's public key. Digital certificates are issued by a trusted certificate authority (CA). Entities acquire a digital certificate by first generating a private/public key pair. The entity generates a digitally signed certificate request that includes a copy of the public half of a key pair and forwards the request to a CA. Once the CA is satisfied that the entity's identity has been verified, it creates a digital identity certificate for the entity by digitally signing the entity's certificate request. Services and others that trust the CA vetting of an entity’s identity validate the CA's digital signature on the entity's digital identity certificate.
In practice, an entity's private key is typically protected by means of a pass phrase used to encrypt it before it is stored in a file or on a removable storage device. Every time the entity wishes to authenticate using its digital identity certificate, it must decrypt the private key to use it. This quickly becomes a cumbersome and impractical task for users. In addition, every time the private key is decrypted for use, it may be temporarily exposed in its unprotected form to opportunities for compromise. It is therefore desirable that entities' long-term private keys be decrypted for use as seldom as possible.

Frequent use of digital identity certificates increase potential for key compromise. Therefore, methods to decrypt long-term private keys as seldom as possible emerged. A proxy certificate is a new certificate generated from a new, short-term private/public key pair, generated and digitally signed by the user using his/her long-term private key [77]. Placing trust in a proxy certificate is a matter of following the signature chain up to the signature of the trusted CA. Therefore, a service may trust the proxy certificate if its user's digital signature validates correctly and the user's digital signature may in turn be trusted if the CA's digital signature on the user's digital identity certificate validates correctly. Alternatively, Kerberos [37] permits network authentication without exposing a long-term shared secret on the wire. Kerberos is based on the key distribution model developed by Needham and Schroeder [53]. A trusted third party, called KDC, holds in confidence the secret keys known by each entity on the network. Each security domain (realm) creates its own KDC. An initiating party conducts a three-party message exchange via the KDC to prove its identity to the contacted party. The initiating party

---

4 Note that anyone with access to the corresponding public key can decrypt the message
5 These secret keys are established out-of-band.
presents a ticket and authenticator to prove it identity to the contacted party. The trusted
third party issues tickets used during the message. The ticket identifies a principal and
establishes a temporary encryption key used to communicate with that principal.

There are also several initiatives that implement a hybrid approach to identifier
representation and management. For example, KX.509 [40] generates X.509 proxy
certificates from an existing Kerberos installation. This allows a Kerberos-based domain
to utilize middleware that expects a proxy certificate representation of identity without
investing in a duplicate PKI infrastructure. The My Proxy project [51] also offers a
hybrid approach to identifiers. Identity is still backed by a long-term secret represented
as a standard X.509 certificate, and the MyProxy server issues short-term proxy
credentials. Whereas identity verification is identical to the process used for user-
generated proxy certificates, long-term credentials are protected similarly to a Kerberos
domain.

Policy

Much like there are several established representations of identity, there are also
several widely adopted representations for access control policy. The most basic
representations are the completely proprietary application specific policies enforced by
application logic and POSIX [64] file permissions. However, applications often wish to
offload authorization processing to the underlying operating system. Therefore, an
operating system usually offers authorization services based on a standard policy
representation. These policies remain distinct from each application, yet can usually
represent the policy needs of the application. The most common representation for policy
at this level is the Access Control List (ACL) [68]. Although many operating systems
use ACLs, including Windows, NetWare, OpenVMS, and UNIX, to enforce security, list
formats, and interpretation are operating system specific. Thus, for policy to be
distributed across heterogeneous systems, a policy designer must understand the syntax
for each specific platform. Further, principals are represented with local accounts and
privileges are assigned directly to the local identity or to local groups.

Alternatively, there are two XML-based policy representations that seek to
abstract operating specific representations of policy. The first of these is XACML. The
Web Ontology Language (OWL) [59], the most recent revision of DARPA Agent
Markup Language (DAML) [15] that is based on the Resource Description Framework
(RDF) framework [45] is an alternative XML-based policy representation. This
framework creates semantic relationships by creating a directed graph of tuples of
(subject, predicate, object) statements pertaining to resources. This graph is a description
of concepts and relationships that exist for a particular resource. However, this
representation restricts the actions that can be authorized to the set directly defined by a
particular ontology. To authorize a combination of actions from multiple ontologies
requires the creation of a new ontology that contains all the desired actions

1.7.4 Identity Management Frameworks

An identity management framework manages not only specific identifiers but also
the characteristics or attributes known for an entity. In fact, the current generation of
identity management systems brings together four major components: directories that
hold the attribute data used during authorization; a management system to add, modify,
and delete the data; a security system that regulates access; and an auditing system that's
designed to ensure company compliance with privacy regulations. [42] These products
are termed provisioning systems and expose proprietary interfaces. With a federated
network identity, network identities that represent a single entity, but are managed by
multiple provisioning systems, can be linked or shared. Liberty Alliance, SAML, and
Microsoft Passport are all frameworks within which multiple identity providers can share
a principal’s identity according to the business relations established between those
providers.

**Liberty Alliance**

Liberty Alliance establishes networked identity management standards and
specifications for organizations to participate in what are called “circles of trust” and
“account federation.” It anticipates federation of diverse emerging identity providers,
each of which may internally employ different identity management systems. These
organizations operate under specific business agreements that dictate how they use the
identities and conduct business. Liberty Alliance strives to provide the framework by
which identities can be federated within a circle of trust. Its ID-FF framework is
designed to work across heterogeneous systems and browser platforms. [57] The Alliance
recommends OASIS' SAML as a way to pass identity information between two sites.
However, "while Liberty is an interop system for connecting identity systems, it doesn't
necessary mean that two participating services will share a particular identity.” [78]
Passport

Microsoft Passport is a protocol that enables users to sign onto many different merchants' web pages by authenticating themselves only once, using a single set of credentials, to a common server. In the Passport model, there are three entities: the client at a web browser, the merchant, and the Passport login server. The login server maintains authentication and customer profile information for the client and gives the merchant access to this information when permitted by the customer. Passport was designed to use existing web technologies, so that clients and servers need not be modified. The protocol leverages HTTP redirects, Java Script, cookies, and SSL. Passport is a proprietary service based solely on Kerberos and functions only on the Windows platform with an Internet Explorer browser.

SAML

As described above, SAML defines a vendor-independent XML data format for representing security information. Within the SAML framework, a SAML Authentication Assertion contains information pertaining to an act of authentication. This assertion can be shared as an identity token across systems. Much like the Liberty Alliance ID-FF framework, SAML targets a heterogeneous environment and builds on the capabilities of existing identity management products.

Commercial Provisioning Systems

No one vendor offers an exhaustive suite of identity management products. But
many vendors are partnering to offer interoperable products that piggyback functionality to provide web access management products, user account-provisioning tools and password management technologies. For example, Business Layers has wrapped its provisioning technology into Netegrity's newest iteration of IdentityMinder [54]. The upgraded identity and access management software includes password synchronization and management, provisioning policies defined by a user's role-based attributes and new auditing, and reporting tools. It currently supports BEA Systems WebLogic [4] and will soon support IBM WebSphere [84] and JBoss [33]. Similarly, Oblix COREid [56] provides a range of identity management features and integrates with most application servers on the market [56]. Other major security vendors provide similar offerings. [73] An effective identity management system incorporates one or more methods of authentication to verify the user, including passwords, digital certificates, or hardware or software tokens. Emerging web access management software--such as Netegrity's SiteMinder [54], Entegrity's AssureAccess [16], RSA Security's ClearTrust [67], Oblix's NetPoint [56], and Entrust's GetAccess [73] allow administrators centrally control user identity and access, enabling SSO through a policy server that grants authorization rights to each application that leverages a particular identity [91]. All of these products are commercial products, with specific platform and environmental requirements, licensing restrictions and infrastructure needs. Within these products, support for SAML authentication assertions as an identity token is becoming the standard thus reducing the constraints on interoperating these products. Unfortunately, most vendors are still focused on identity federation and SSO and rely on internal infrastructure to provide authorization based on managed (or shared) identity.
1.7.5 Security Toolkits

Security toolkits facilitate the use of standard security mechanisms by abstracting the details of implementation or message formats into an object model that is familiar to the developer. Typically, these toolkits provide an API or object model in a specific language or for a specific platform for a particular type of security information such as XML Digital Signature, WSS, SAML or XACML. Such toolkits greatly ease the authorization developer’s burden by shifting their focus to authorization specific logic from the minutiae of generating data packets or on-the-wire messages.

XMLDSig

There are several toolkits available that implement security standards for XML that include XML-Signature Syntax and Processing and XML Encryption Syntax and Processing. All implementations of the XML Signature Syntax and Processing standard require the implementation and use of an entire set of XML technologies, including XML parsing, XSLT, and XML Schema tools. Further, each of the available toolkits supports the generation and verification of the three signature types: enveloping, enveloped, and detached. Finally, all the toolkits support the XML Signature – Syntax and Processing W3C Recommendation dated 15 February 2002.

The Apache Software Foundation’s XML security [89, 48] project provides both a Java and C++ implementation of the recommendation. However, there are currently no standard APIs available that implement the recommendation; therefore the
implementations do not claim comply with any standards currently under development.
The library allows for the creation and verification of digital signatures expressed in XML as well as signing and verifying XML content. There are several Java library specific considerations that must be addressed when using this software. However, options for resolving potential problems are clearly documented both within the distribution and in generated runtime exception messages.

IBM also offers an XML Signature implementation. [90] This Java toolkit requires J2SE v1.2 or later together with some specific versions of external XML libraries. The library supports the generation of XML signatures via templates, which are incomplete signature documents. A template has no <KeyInfo> element and no content of the SignatureValue element and DigestValue elements. It presumes that the user has a key-pair when generating a signature, but no such requirement exists for signature verification.

Finally, the IXSIL library (30] is offered by the Institute for Applied Information Processing and Communications (IAIK). IXSIL is a toolkit that enables Java developers to easily integrate the creation and verification of XML based digital signatures into their applications. In addition to the required and recommended signature processing functionality, IXSIL permits the developer to specify the XML parser and XPath engine of their choosing. Further, it allows the developer to specify proprietary algorithms for signing and digesting as well as defining custom key information.
WSS

Most of the major contributors to the initial WSS and now-evolving OASIS specifications offer a WSS toolkit. Some toolkits leverage the functionality of a specific SOAP engine, such as the WSS4J toolkit. [28] This toolkit is implemented as an Axis handler and directly leverages the Axis API to provide WSS functionality. Other implementations of WSS provide the functionality as an add-on to their larger web-service or security offerings. Other vendors that provide such a toolkit include Phaos [63], Vordel [82], Microsoft [46], and Verisign [81].

SAML

Most SAML toolkits are part of a larger identity management or identity federation offering that provides SAML compliance via extensions to previously proprietary functionality. Typically SAML support is integrated only to provide SAML compliant messages representing internal product functionality to be placed on the wire. The SAML messages are then manipulated with proprietary functionality. Further, most web service management products support SAML as a token format to consume as an SSO credential but rely on SSO products themselves to generate the assertion to be consumed. Opensaml [58] is the only open source toolkit available that fully supports the SAML specification. It provides a Java API to generate SAML protocol packages, statements, and assertions as well as consumes SAML messages and provides access to the data therein via an object model rather than via direct manipulation of the XML construct. Other vendors that support the SAML specification typically expose the contents of SAML messages via their internal APIs. Thus, that information feeds directly
into their product functionality that leverages that proprietary API. These products support a variety of operating systems and languages, but are dependent upon the support offered by the larger vendor platform. Vendors with such an offering include Novell, Netegrity, RSA, Sigaba, Oblix, OpenNetwork, and Sun Microsystems [73, 63, 56, 67, 54].

**XACML**

There are two XACML toolkits currently available. Both of these toolkits supply the functionality specified in Version 1.1 of the XACML. However, this specification purposely leaves unspecified many critical features of an access control decision process, such as how to locate applicable policies. These toolkits address these features in the most basic manner. The first toolkit, sunxacml [75], is an open source Java implementation developed at Sun Microsystems Labs that requires J2SE version 1.4 or greater. Sunxacml offers full support for parsing both policy and request/response documents, determining applicability of policies, and evaluating requests against policies. All standard attribute types, functions, and combining algorithms are supported. The toolkit also provides APIs for writing new retrieval mechanisms used for finding policies and attributes. Sunxacml does not provide PDP or PEP implementations; rather it provides the APIs necessary to perform those tasks as well as creating, encoding, and validating policies within a defined object model. However, the distribution does not offer any support for locating and loading policy files into the PDP or for network communications. The second is a suite of products developed by jiffySoftware [34]. The suite contains three tools all implemented in C++ and supported on Linux and Win32.
platforms. The policy tester is a command-line tool that can perform syntax and type checking on XACML policies and requests as well as execute requests against a policy. JiffyXACML passes the entire XACML conformance test suite. The policy engine provides the access control decision engine features available in the policy tester as an embeddable component for an application. The policy server is an access control enforcement environment provisioned with the features of the policy tester that can be deployed in web-service and grid computing environments via the XML-RPC interface.

1.7.6 Alternate Authorization Approaches

To this point, all the functionality highlighted in this section does not focus specifically on providing access control in a distributed environment. It focuses instead on underlying problems such as message security, identity or exposing the features of a standard as an object model. In addition to these raw tools, there are several systems that approach the problem of distributed authorization from different angle. These systems vary in their representation of identity, policy, and authorization decisions or continue to functionally integrate authorization decision and enforcement.

KeyNote

The KeyNote Trust Management System [7] defines a trust management system to contain five basic components: a language to describe actions, principal identity, a specification for policies, a language to delegate authorization, and a compliance function. Thus, semantic application differences won’t affect security configuration, even with distributed policies. However, KeyNote assumes invocation via a function call
interface within an application. This assumption limits the potential platform
heterogeneity supported by the system. The call returns one value from a specific
ordered list of possible responses pre-configured by the application. Thus, KeyNote
cannot be independent of and transparent to the application.

CAS

resource providers to specify course-grained access control policies in terms of
communities as a whole. Resource administrators can thus delegate fine-grained access
control policy management to the community itself. Therefore, resource providers
maintain ultimate authority over community access to their resources but must cede
control of intra-community decisions. Further, although administration is partitioned
between community and resource administrators, knowledge of resources, user identities,
community access rights, and group memberships must be established before
authorization may occur.

Akenti

Akenti [49] extends the CAS community model to support multiple stakeholders
that may impose use conditions on a particular access control request. Akenti provides a
way to express and to enforce an access control policy without requiring a central
enforcer, and administrative authority. The system’s architecture is intended to provide
scalable security services in highly distributed network environments. Attribute and use
case conditions are encoded in digitally signed certificates that are collected during the
authorization decision process (pull model) and evaluated according to the Akenti policy
language. Although the pull model for collecting data facilitates system distribution, the proprietary policy language developed for Akenti requires that any participant use specialized, rather than standard, data formats, and APIs.

**Permis**

The PERMIS [11] system focuses on both authentication and authorization with the use of identity and attribute certificates to represent both the mutable and immutable model of a user. While implemented with XML and Java technologies, the access control API, as well as mechanisms to allocate privileges, is currently proprietary. Further, the reliance on a pull model to collect attributes and the X.509 attribute certificate [18] standard to represent attribute values represent assumptions that are not necessarily valid across heterogeneous environments.

**Prima**

The PRIMA system [44] provides mechanisms that focus on the management and the enforcement of fine-grained access rights. The solution employs standard attribute certificates to bind rights to users (or their surrogates) and enables the high level management of such fine grained privileges which may be freely delegated, traded, and combined. However, enforcement relies on POSIX [64] operating systems extensions that extend standard file permissions and regulate resource usage through access control lists.

**KAoS**

The KAoS system [1] together with its associated policy language (KPO) and
policy administration tool (KPAT) provides an ontology-based system that describes relationships between objects in a domain to support authorization. The KPO defines basic ontologies in terms of classes and properties. Applications rely on KAoS for an authorization decision obtained via an Enforcer associated with each protected resource. An enforcer intercepts requested actions and checks against known policy to return the appropriate decision. KAoS directly manages policies distribution by applying the Java Theorem Prover according to the ontology. The major focus of the system is reaching an authorization decision and the authorization representation feeding into the decision. However, it employs proprietary communication between its components and deployment of the system is end-to-end. Thus, it is tightly coupled with the identity and enforcement mechanisms.

Shibboleth

Shibboleth [17] provides a web-based authentication and authorization system. The primary use case is securing interaction between higher education sites, though it is generally useful for any environments that must work across domains of trust. The system works entirely within the scope of a web browser. After the browser issues the initial resource request, a series of exchanges, executed by the Shibboleth implementation, between the target site and the user's site verify the user's identity, gather attributes, and perform the access check. Despite this flexible attribute management system, actual policy decisions are ultimately made using .htaccess files in an Apache module, thus limiting the type of resource that may be protected.

Existing authorization systems do not individually span the distributed
authorization problem space. Several of the available systems retain platform or resource specificity requirements. Others require or assume specific identity credentials, enforcement mechanisms or couple the decision, and enforcement mechanisms. Moreover, although enforcement of an authorization decision or information stores for data consumed during the authorization decision may be distributed, there no hooks to distributed policy management or decisions. Finally, many of these systems do not provide online capabilities for authority discovery and interaction nor mechanisms to exchange data in-band.

2 Cardea – A Departure from Traditional Approaches

Cardea utilizes state of the art technologies in a robust, flexible and scalable authorization framework. Cardea departs from traditional authorization approaches in four areas. First, Cardea does not define access control policy in terms of specific identifiers, thereby allowing security administrators to focus on policy semantics. Second, Cardea decouples the authorization and authentication functions so that portable identity credentials need not be statically mapped to local identity representations. Third, Cardea separates authorization decision and enforcement so that an authorization decision can be reached independently of the specific features of underlying subsystems to enforce authorizations. Finally, Cardea specifies critical authorization interfaces rather than their

---

6 For example, a compute resource may support the concept of a process group owner. Determining the authorization for a particular process may attempt to set the process group owner to an appropriate value. Success or failure of that set-owner operation determines, in part, the authorizations granted to the process. However, a specialized instrument may not recognize the concept of a process owner. Thus, without
implementations, thus abstracting away the differences among underlying mechanisms.

2.1 Key Design Issues of Cardea

Several key design issues arise from the Cardea authorization model. These issues impact the solution architecture that adopts this model. These critical dimensions are: choosing a communication paradigm; establishing common semantics; selecting an authorization algorithm; and performing authority binding. Ignoring any of these issues would force as many, albeit different, assumptions as traditional authorization decision architecture.

2.1.1 Communication paradigm

The communication framework for Cardea must represent the following conceptual transaction participants: the requester; the subject of an authorization decision request; and trusted authorities. Additionally, the framework must recognize the following roles: guarantor of the information, the authorization decider, and the enforcer are conceptually distinct roles. The communication framework must support content and meta-data regarding any of the participants to be exchanged between any entities assuming one of the roles.

2.1.2 Information representation

Cardea must encourage common semantics across organizational boundaries.

decoupled decision and enforcement functions, there would be no way to establish authorization for this pair of resources in a single transaction.
These semantics must not be limited to an established set of identified entities, but rather must process any relevant characteristic of a transaction participant. Further, the selected representation must make meta-data, such as data-type or scope, available to clarify the semantics of shared information. Finally, the selected representation must be transformable to support multiple internal mechanisms that process information.

2.1.3 Implemented authorization algorithm

To successfully support a global view of an authorization decision, Cardea’s authorization algorithm must demonstrate several critical characteristics. First, the algorithm must indicate what information is required for a particular decision and how to provide that data to the decider. The decision process must be independent of the actual values on which it operates. Finally, the algorithm must not perpetuate a conceptual centralization of decision power.

2.1.4 Authority binding

Additionally, the architecture must support any combination of local and remote users/resources. Therefore, authoritative sources of information must be located and queried in-band to a transaction. The solution must address how to identify the appropriate authority to contact, how to communicate with that authority, and what to communicate with that authority. Further, these decisions must be based pre-established trust relationships or specific transaction data.

2.2 Initial Prototype Features

The initial Cardea prototype proves the viability of this model of dynamic authorization. The prototype architecture establishes a communication paradigm and
information representation based on the features of several web service standards: SAML asserts entity attributes and authorization decisions; XACML standardizes access control policy evaluation; SOAP defines the messaging capabilities; and WSS provides a message security layer atop SOAP. The prototype also defines standard attribute definitions to provide a common semantic basis for the decision process. Finally, the prototype demonstrates how to build an attribute authority from existing provisioning infrastructures.

Most critically, Cardea establishes a model of trust for cooperating system components. The model of trust addresses four critical requirements of a distributed authorization system. First, the model identifies integration strategies for existing authentication schemes. Second, it identifies how to identify the authority for a particular access control decision. Then, it defines how to interact with trusted authorities. Finally, the model provides the mechanisms to spontaneously establish new relationships of trust, when required by a specific transaction. Cardea strives to maximize flexibility and autonomy within this model of trust by supporting the various interaction modes commonly found in existing infrastructures.

### 2.3 Architecture

Cardea is currently implemented as a set of independent functions exposed as services. The majority of these services support the “Evaluate” and “Decide” phases of authorization, or on sharing information between phases. The backing implementations for each service are implemented in Java [32]. Java was selected for its well-defined and open XML/SOAP processing capabilities, its platform independence, and the wide
availability of specifications and compliant servers.

Figure 9. Cardea Architecture

Conceptually, a single security domain contains several internally accessible authorities and a gateway authority that accepts requests from external domains and routes them to the appropriate internal authority. In Figure 9, the PEP\(^7\) intercepts a request from user U1 to access resource R\(_{a1}\). Each PEP is configured to contact an authority it trusts to execute an authorization decision and then enforce that decision. A domain also contains two types of authorities. The first authority type evaluates authorization requests. These authorities are titled Policy Decision Points (PDP). The second authority type generates attribute assertions about domain users and resources. These authorities are named Attribute Authorities (AA). In Figure 9, the PDP contacts

\(^7\) The PEP may or may not be co-located with the resources it protects.
the AA to request certain attributes for U1. However, both a PEP and/or PDP may contact an AA to obtain necessary attribute information. Finally, a domain may establish an Information Service (IAS) to store PEP and PDP location lookup information for the domain. The IAS also contains lookup information on external trusted gateways and internal AAs. Finally, the gateway is itself a PDP. It decides when to forward an external resource authorization request to the appropriate PDP or AA. To make this determination, the gateway can query the IAS to determine which internal PDP is authoritative for the request. In Figure 9, any request from U1 to access a resource in Domain B will pass from the Gateway PDP in Domain A to the Gateway PDP in Domain B. The Gateway PDP in Domain B determines which PDP within Domain B should evaluate the request.

Figure 10. Internal PDP Architecture
Internally, each Cardea PDP is implemented as an XACML Context Handler and Policy Authority\(^8\). The PDP also acts as an AA client, issuing attribute queries and accepting assertions in return. The PDP is also configured to contact an IAS in some unspecified manner. See Figure 10 for a visual depiction of this breakdown.

![Figure 11. Cardea Message Structure](image)

Each PDP function is exposed as a location-transparent web service. Message forwarding and delivery relies on a SOAP messaging layer. Native XACML and SAML protocol messages are opaque message payloads. WSS headers guarantee these payloads. Figure 11 illustrates the structure of a single message. An XML firewall processes these security headers, verifies attached signatures and forwards the opaque payload to an appropriate authority. A gateway authority must conceptually encompass these capabilities while individual organizations decide how to provide the necessary functionality internally.

\(^8\) In the XACML specification, a policy authority is termed “Policy Decision Point.” A different name was adopted for this component in Cardea to avoid confusion with the Policy Decision Point as defined by SAML.
2.3.1 Platform implementation

We selected open source components to establish the underlying SOAP and transport platforms. These components adhere most closely to open standards and specifications. Support for SOAP-based communication comes from the Java reference implementations of the API for XML messaging [31] and utilizes the Apache Axis [88] architecture as an engine to transmit SOAP messages atop the http/https communication protocols. The Jakarta Tomcat Servlet [76] engine was selected as the application server. Tomcat also contains an embedded http server that natively handles http specific processing. Therefore, a standalone web server was not required. Figure 12 illustrates how the platform components work together to consume a single message.

Handlers are chained together to implement the correct XML firewall capabilities
for each web service endpoint hosted by the Axis engine. The handlers process the WSS SOAP headers using the WSS4J libraries to generate and verify security header elements. These handlers use the Apache xml signature [89] library for XMLDSig processing.

After each message passes through the appropriate handler chain, Axis extracts the opaque SAML or XACML payload and forwards the content to the appropriate SAML or XACML-aware authority. Cardea’s SAML-aware authorities were built using the opensaml [58] toolkit while the XACML-aware authorities were implemented using the sunxacml [75] library.

2.3.2 Web services

Adopting a web-service based SOA displays several characteristics that are attractive to the system architecture. Most importantly a web services model shields the user of a service from any details of the service implementation. Thus, the radically different backing functionality of local and gateway PDPs remains transparent to the user. A web service model also offers integrated XML processing. Thus, the system can use the wealth of XML tools and processors currently available. Further, a web-service approach provides a framework to exchange both message payload and meta-data. This simplifies control channel requirements that must be established across COIs before collaboration can begin. Finally, a SOAP engine can directly pass opaque messages to backend handlers without requiring computationally expensive transformations to marshal messages into native SAML and XACML message formats.

2.3.3 Standards conformance

As discussed in Section 1.6, Cardea’s overall architectural approach relies on
open standards conformance. Conforming to the SOAP specification allows the messaging framework to remain completely transparent to application processing components. SOAP also permits arbitrary amounts of meta-data to be added to a SOAP envelope. Further, the WSS specification defines a standard binding for security information to SOAP messages. Therefore, an XML firewall can process WSS headers before passing the payload to the service endpoint. SAML conformance allows external representation and exchange of data between SAML-aware entities [14] regardless of internal security architectures. [61] Finally, XACML defines a general-purpose mechanism for expressing and evaluating the access control policies [86] independently from the method by which they are obtained and formatted for the PA. Therefore, the SAML specification governs the collection and expression of these attributes and XACML generates a decision for virtually any combination of such attributes.

2.3.4 Modes of operation

Cardea supports multiple PEP interaction modes to avoid over-stipulating credential or enforcement mechanisms. With the push mode, a principal generates an AuthorizationDecisionQuery (ADQ) directly, with a self-representing subject. The principal contacts the PDP directly for an authorization decision that is forwarded to the PEP as an authorization credential. In the pull mode, the PEP receives a request in some unspecified manner from the principal. The PEP selects the PDP to evaluate the request and generates an ADQ whose subject is the requesting principal. The PEP receives the authorization decision directly from the PDP. The final supported mode is the hybrid mode. In this mode, the principal communicates with the PEP but like the push mode, provides some indication as to the PDP to contact for a decision. Afterwards, as in the
pull mode, the PEP generates an *ADQ* whose subject is the requesting principal. Finally, the PEP receives the authorization decision directly from the PDP.

Cardea also supports the push, pull, and hybrid modes of operation for an attribute request (*AR*). A principal can request attribute assertions prior to generating the *ADQ* and attach them to the *ADQ* as evidence to use during evaluation (*push*). If no attributes are directly attached to the request, the principal may attach meta-data attribute assertions that identify the attribute authorities to contact for requesting user and resource attribute information during the decision information (*hybrid*). In either case, the deciding authority may request attribute information independently during the decision evaluation (*pull*).

### 2.4 Pre-requisites and Assumptions

Although the system minimizes the amount of negotiation and configuration that must occur prior to collaboration, several aspects of XACML and SAML semantics remain to be profiled.

1. Local access control policies must be defined according to characteristics of salient user-resource-action-environment combinations. See Appendix F for a simple example of an XACML policy file.

2. Each authority and PEP should be identified by a unique URI [6]. This naming scheme requirement supports:
   - The identification of service endpoint targets at the web service level
   - The SAML constructs that utilize URI identifiers
3. An AA must be able to provide each attribute value to a qualified requester as a SAML Assertion. There is, however, no inherent restriction on how attributes are maintained internal to an attribute authority.

4. Authorities must be SAML and XACML aware and must understand defined attribute profiles.

5. XML firewall functionality can digitally sign message payloads.

6. Any process that generates an AR or authorization decision request (ADR) must be able to correlate each request with the resultant response via SAML protocol identifiers.

7. Each ADR must comply with the profile defined in Appendix I.

### 2.5 Information Representation

Selected information representations should encourage scalability, flexibility, and interoperability while establishing common semantics to implement locally. To this end, Cardea defines SAML profiles to represent a requester, the subject of a request, attributes, and meta-data in a standard manner. These profiles are not mutually exclusive. For example, a SAML Subject will represent an authority in any authority meta-data attribute assertions while a URI will represent that authority as the issuer of attribute statements. Additional rules that do not rely on SAML profiles for information representation are also discussed in the following subsections.

#### 2.5.1 Message Sender

Based on its initial deployed in a PKI environment, all Cardea entities, including host resources and authorities, possess X.509 identity credentials. This credential represents the identity of a requester and the associated public key is used to verify the
message signatures. The SOAP firewall evaluates the digital signature presented in the
SOAP:Header to evaluate the trustworthiness of the message sender. Recall that a
message payload is opaque to the XML firewall. Therefore, the firewall can enforce no
semantic correlation between the sender of a message and its content. The firewall
performs three verifications on each sender representation before passing it to the
intended endpoint. The first verification ensures the sender generated the request. The
second verification confirms the request was not altered in transit. The final verification
determines whether the sender is permitted to make the request.

2.5.2 SAML Subject

An access request may be made on behalf of a principal and not by the principal
directly. For example, a PEP may contact the PDP when a principal requests access to a
protected resource. The PEP is the message sender whereas the principal is the subject.
Therefore, the subject may differ from the sender. To provide flexible integration with
existing provisioning systems, Cardea supports three profiles of the SAML subject syntax
to represent these end entities. These profiles are defined in Appendix H. A SLA
agreement between COIs specifies which of the profiles may be encountered in
exchanged messages. However, decision and attribute processes manipulate the SAML
subject structure directly. Therefore, the actual profile selected by a domain does not
alter the decision process.

2.5.3 Authority

There are several SAML-defined constructs that represent an authority with a
URI. XACML also relies on a URI representation of attribute authority identity.
However, neither specification stipulates how to craft a URI to identify a specific
authority. Thus, Cardea defines a standard transformation for converting a host identity into a URI representation. This transformation adopts the following two rules:

1. The fully qualified domain name for the host of authority is appended to the XACML subject-identity URI.
2. These URI components are separated with the separator symbol, “:”.

Therefore, an authority located on host myhost.foo.bar.com would be identified by the following URI:

```
urn:oasis:names:tc:xacml:1.0:subject:subject-id:myhost.foo.bar.com
```

### 2.5.4 Service Endpoint

Authorities are targeted as service endpoints for Cardea messages. However, there may be several authorities, and thus multiple potential endpoints, hosted on a single resource. Therefore, Cardea must establish how to target a particular message to a specific endpoint rather than a specific host. As message content is opaque to SOAP processing, any authority identity contained within the message itself cannot be used to identify the targeted endpoint. Moreover, the requester knows only the gateway PDP identity when requests cross organizational boundaries. Although an SLA defines the service endpoints that serve as gateways into an organization, configuring the identities of known external gateways and defining forwarding mechanisms from the gateway to internal authorities were deemed bootstrap problems that rely on the SOAP platform selected. Therefore, these are purposely left unspecified.

### 2.5.5 Intermediaries

In a distributed model, each message may pass through several hops between its origin and final destination. Each “hop” is termed an intermediary. Some authorization
decisions want to consider not only the originator and recipient of a request, but also all
the intermediaries that could potentially touch the message. There is no native SAML
mechanism to convey the intermediaries through which a request passes. Therefore,
representation of an intermediary occurs only within WSS headers. The identities of any
intermediaries through which a message passes can be attached to the messages as
additional security signatures on the data or as security tokens inserted as meta-data in the
headers. Although not mandated, this ability to represent intermediaries within a
message provides flexibility for each domain to configure its XML firewalls to verify or
disregard intermediary information. Although intermediaries are not represented in
current Cardea functionality, the design leaves a clear pathway to support the inclusion of
intermediary identities in a future release.

2.5.6 SAML Attributes and Attribute Meta-data

SAML Attributes represent the characteristics of entities within the system. An
AR generates AttributeDesignators, packages them within a SAMLAttributeQuery
(SAMLAQ) and then considers returned assertions for the duration of the authorization
decision. Appendix B contains an example of an SAML AR constructs generated by
Cardea. Although the AttributeDesignator construct is intentionally generic [14], there
are several existing attribute profiles, such as LDAP/X.500 attribute profile that establish
common attribute definitions and semantics. See Appendix D, Appendix E, and
Appendix F for attribute definitions that carry standard semantics for Cardea processors.

Exchanged attributes are not limited to those pre-defined by profiles. Additional
profiles that define common identity, semantics, and metadata for a set of attributes may
also be established as part of a SLA. Appendix I defines how these facets must be stipulated in an SLA.

According to SAML syntax, an authority must package $\textit{SAMLAttributeStatement}$ into a $\textit{SAMLAssertion}$ before placing the assertion into a $\textit{SAMLResponse}$ structure. Each $\textit{SAMLAssertion}$ may contain an $\textit{Advice}$ element. In Cardea, the $\textit{Advice}$ element contains the meta-data for all statements wrapped within that assertion. The meta-data value that applies to a particular attribute can be found by traversing the following SAML Assertion XML pathway: $\textit{SAMLAssertion} \rightarrow \textit{Advice} \rightarrow \textit{SAMLAssertion} \rightarrow \textit{SAMLAttributeStatement} \rightarrow \textit{Attribute} \rightarrow \textit{AttributeValue}$. There may be any number of proprietary attribute meta-data statements contained within a single $\textit{Advice}$ element. However, Cardea defines several standard meta-data attributes. See Appendix C for profile rules for $\textit{SAMLAttributeStatements}$ that carry attribute meta-data. In addition to specific profile rules, each meta-data statement must apply to every attribute statement represented within the enclosing assertion. Therefore, if an authority returns multiple attribute statements that vary according to meta-data, they must be grouped into separate assertions according to their meta-data.

### 2.5.7 Authority meta-data

Authority specific attribute statements are exchanged in self-signed assertions. The statement subject represents the authority as a SAML subject construct. The system assumes that the requester knows the identity of the authority by some bootstrap or out-of-band mechanism\(^9\). Each assertion that contains authority meta-data statements is

---

\(^9\) The requester must still verify the authority signature.
considered valid for the lifetime specified within the containing assertion. Each authority meta-data statement conforms to the same attribute profile for attribute statements for any principal. However, several attributes are defined that make semantic sense only the subject is an authority. Appendix D deals with these attributes.

2.5.8 Standard mappings between SAML and XACML representation of information

XACML defines a specific attribute representation that dovetails with its policy representation. Therefore, Cardea defines rules to govern the transformation of SAML attribute statements into XACML attribute designators\(^\text{10}\). These rules transform data contained directly in a *SAML Attribute Statement* as well as meta-data that may be represented elsewhere in the containing *SAML Assertion*. This functionality presumes a correspondence between the attribute identities used in the XACML and SAML representations of each logically equivalent attribute.

2.6 Trust Model

2.6.1 Roots of trust

As an authorization system, Cardea relies explicitly on trusted information. Certain sources of such information sources are explicitly trusted in all circumstances. These sources are contractually established between security domains. RFC 2904 identifies the need for bilateral service agreements, in the form of formal contracts or Service Level Agreements, between pairs of security domains involved in an authorization transaction. A grid constitution is an example of such an organizational

\(^{10}\) Version 2.0 of the XACML specification set contains a SAML attribute profile that standardizes such a transformation process.
level agreement. RFC 2904 additionally identifies the need for agreements between a user and the user’s security domain. Similar relationships identify trusted authorities internally to a particular security domain. Figure 13 illustrates these relationships. These static relationships form the roots of trust within the system and are the path elements through which dynamic relationships between arbitrary user-resource pairs. For example, PEPs are directly configured with the identities of the PDP(s) to trust. One of these PDPs serves as a gateway PDP that is configured with the identities of the gateway PDPs in remote security domains with which a SLA agreement exists. This allows a local PEP to present (via the gateway PDPs) an authorization request to any remote PDP that is trusted even without specific knowledge of that PDP.

![Figure 13. Service Agreements to Establish Trust](image)

2.6.2 Sign SOAP payloads
According to section 8.4 of the WSS specification, even if the syntax and verification of the signature attached to a message are correct, an XML firewall must still determine if the specific signer presented is trusted. When a PEP propagates the request on behalf of the principal, the PEP signs the message directly. Otherwise the principal (potentially proxied) that generates the initial ADQ signs the message. These signatures permit an XML firewall to verify the message sender. Similarly, the recipient of an authority’s message must determine if the generating authority is trusted. Note that this signature-processing model does not preclude the use of transport level security mechanisms such as https or GSI. [10]

2.6.3 Signed SAML assertions

As the SAML protocol is designed to support bindings to multiple messaging and transport protocols, it defines SAML-native mechanisms to guarantee assertion content independently of any guarantees provided by the underlying messaging infrastructure. By verifying the signatures on each assertion, the recipient can independently establish trust in the authority that generated the assertion independently. Establishing this trust follows pattern similar to verifying the signature on a SOAP payload.

2.7 Supporting the Phases of Authorization

Cardea functionality implements several phases of the authorization model. Recall that conceptually, an authorization transaction can be broken into four phases: requesting a decision, evaluating a decision, making the decision, and enforcing the decision. The following subsections identify several critical design points pertinent to each of these phases.
2.7.1 Requesting an Authorization Decision

Mode of Operation

The selected mode of operation affects the interaction flow between requester and recipient. When operating in the push-messaging mode, a process initiating an authorization request collects attribute information to forward with a decision request. This consists of building SAML AttributeQuerys and including each attribute assertion received in response within the ADQ. This requires maintenance of state to complete the Attribute querying process and transform the results appropriately into the Advice format. When operating in a hybrid-messaging mode, a process initiating an authorization request must build an attribute assertion containing the identity of any attribute authority it wishes the PDP to contact for attribute information. When operating in the pull-messaging mode, a processing initiating an authorization request builds an ADQ without any additional attribute or meta-data information, leaving all related decisions to the PDP.

Selecting the Appropriate Authority

Regardless of the flow of interaction, each request must be targeted to an authority. When a transaction involves a user and resource within the same security domain, there is no requirement to consult the SLAs in place with any remote security domain. Alternatively, any request for a resource not governed by the same security domain as the requester should be considered remote. Each authorization request for such an external resource should target a PDP that also considers the SLA between the local and remote domain.
Inside organizational boundaries, the identity of each PDP is available via OOB configuration. Security administrators are free to choose whether the information that directly points to the PDP or may reference local information services which report the appropriate PDP for a particular resource. Any changes to this configuration affect only local entities. Out of band information provided to a requester identifies a gateway PDP to contact for access to any resource external to the local security domain of the requester. The gateway PDP internally manages any additional location, lookup and communication details for interaction with remote PDPs, abstracting these details for the requester. Each gateway PDP need know only how to contact the gateway PDP for any remote domain. The gateway PDP for that domain will similarly abstract the details of communicating with any additional PDPs within that domain. Mapping the resource to its governing PDP uses the URI naming scheme. Moreover, the gateway PDP can utilize this resource-naming scheme if it incorporates the identity of the external domain. Otherwise, the SLA should address the established approach for resource to domain mappings.
Once the appropriate PDP is selected, the requester may wish to discover certain metadata characteristics about the PDP. For example, the requester may request metadata attributes, as described in Appendix D, that affect the format of either an attribute or a decision request forwarded to that authority. Cardea re-applies the SAML protocol by treating the authority proper as the subject of its own configuration data. This authority is
deemed authoritative on its own meta-data. By definition, a trusted authority is already a root of trust in the system. Therefore, the recipient of each meta-data assertion must only verify that the signature corresponds to a trusted authority and matches the subject of the authority meta-data statements carried therein.

The requester may need to know what meta-data the authority makes available. As per the SAML specification, an attribute query containing no attribute designators is interpreted as a request for all known attributes for that subject. Thus, a requester can create an attribute query whose subject is the authority in question without including any specific attribute designators and then forwards the query to that authority. Reapplication of the attribute request paradigm provides a flexible mechanism to determine the available meta-data for a particular authority.

Packaging request

After collecting all the information needed to build the authorization request, the requester must appropriately format the information as a SAML ADQ. Appendix K specifies the rules governing the creation and format of an ADQ.

Passing the request to the Authority

There are three general scenarios by which a request can be forwarded to a selected authority. Of the three models, one includes obvious implementation difficulties. That model is described here for completeness and to illustrate the considerations involved in selecting an appropriate model. The two remaining models vary in their approach to determining who is ultimately responsible for presenting an
authorization credential to the PEP. Scenario A, depicted in Figure 15, delegates this decision to the PEP. Therefore, the principal communicates only with the PEP and is ultimately unaware of the PDP’s existence. Adoption of this model dictates several PEP and PDP characteristics. First, the PEP knows which PDP to contact. Second, the PDP returns the authorization credential directly to the PEP, never communicating directly with the principal. In fact, the PDP’s only knowledge of the principal is as the subject of the authorization request. Finally, the PEP must provide some mechanism independent to the authorization process to provide the principal with any authorization enforcement related data. Note that the PEP must also establish a unique identifier for the decision with the principal who has no hook into the SAML exchange between the PEP and PDP.

**Scenario A**

![Diagram of Scenario A](image)

Figure 15. PEP communicates directly with PDP

Scenario B, depicted in Figure 16, presumes that the principal contacts the correct PDP. In this scenario, the PDP forwards the authorization credential directly to the enforcing PEP. Adoption of this model presents three problematic characteristics. First, each principal must know how to locate the governing PDP. Then, the PDP must be able to determine the PEP that will enforce the decision. Finally, the solution requires
redundant messages sent to multiple recipients to provide the requester and PEP with decision information.

Fig. 16. PDP receives request from principal and forwards decision to PEP

Scenario C, depicted in Figure 17, differs from the second as the PDP returns the authorization credential directly to the requesting principal. The principal is then responsible for forwarding the credential to the appropriate PEP. Therefore, the principal communicates with both the PEP and the PDP. With this model, the principal determines which PDP to contact for authorization. The enforcement of the decision occurs only when the principal forwards the credential to the PEP. Thus, there must be sufficient information in the authorization credential for the PEP to verify trust in the deciding PDP. This information may take the form of assertion signatures. It may also use two SAML specific constructs: the Issuer value of an assertion that opaquely identifies, in URI form, the authority that issued the assertion and the AudienceRestrictionCondition element which specifies the intended audience, in URI form, of the assertion. For example, the audience may represent the PEP identity. Finally, the principal must
ascertain the appropriate PEP to which the authorization credential should be presented.

This scenario most closely implements the push messaging sequence as PEP only enforces a decision when provided by the principal.

**Scenario C**

![Diagram](image)

**Figure 17. Principal requests authorization PDP and forwards credential to PEP**

**Passing requests thru an XML firewall**

Cardea treats an authorization transaction as a single conceptual processing operation that spans both SOAP and payload-specific processes. Thus, the \textit{SOAPActor} attribute of the WSS header, which identifies the intended recipient of that header, reflects the PDP rather than an XML firewall. The approach allows a security domain to omit security or adjust filtering based on requester identity without impacting the message format. Cardea mandates the presence of the \textit{SOAPActor} attribute, with a value reflecting the identity of the targeted authority to support future inclusion of additional message processing such as encryption or compression.
Given that the request and response handler logic provides signature verification and generation capabilities, the handlers must have access to the appropriate key. Fortunately, the WSS header carries the certificate containing the key to use during signature verification of an incoming request. This certificate is validated in an implementation specific manner, such as certificate chains or OCSP. During response processing, the XML firewall must have access to the key pair needed to sign the response and to include the verification credential in the WSS header of the response message. Figure 18 depicts the signature generation and verification processes that are applied to a single authorization transaction. Several important error-processing rules are outlined in Appendix L.
2.7.2 Evaluating an Authorization Decision Request
Evaluation focuses on the extracted SAML payload, which is processed according to its SAML content. A chain of interceptors implements the processing supported by an authority. When a message arrives, the request is immediately passed to the head of the interceptor chain. Each interceptor examines the message and determines if it can process the message according to its processing rules. The first interceptor to accept the message will control the processing for that message. If no interceptor known to the authority accepts a message for processing, the authority rejects the message and returns the SAML status code `Responder` with a secondary status code of `RequestDenied`. Currently, there are three interceptors defined: one for processing signed requests, one for processing unsigned requests and one for gracefully handling messages that do not comply with an appropriate protocol version. The final interceptor categorically rejects the message and returns the SAML status code `VersionMismatch` without a secondary status code.

Each interceptor is configured with multiple validators that contain distinct validation rules to apply to a message.Interceptor configuration dictates the application order of validators. Therefore, the implementation attempts to order the validator list from least to most specific. These validators may reject a request for a variety of reasons. If a validator rejects a message, the interceptor returns the SAML status code `Responder` with a secondary status code of `RequestDenied`.

**Information collection**

Once a request has passed through all the configured SAML validators, the Authorization Authority collects the information to provide to the XACML evaluation
function. The authority must then transform any attribute values it received within the Evidence of a SAML request into the correct XACML format. The rules for this transformation were outlined in Section 2.5.8. To minimize the set of attributes included in the XACML request context, a custom interface was built into the SAML Authorization Authority to report the attribute identifiers expected within each <xacml:RequestContext> for a particular resource. This discovery service is named the Policy Profiler. Initial Policy Profiler functionality directly maps resource identifiers to the set of subject attribute identifiers considered by the policy governing that resource. If reporting these attribute identifiers significantly reduces the number of attributes that must be collected, it can result in a significant efficiency boost over blindly collecting all available attributes to include within each <xacml:RequestContext>.

If the Authorization Authority does not find the relevant attribute values within the Evidence of a request or chooses to disregard this information, it must collect all the attribute information to pass to the XACML decision process. Thus, the PDP constructs SAML attribute queries for the request subject for each attribute identifier reported by the Policy Profiler. Locating the appropriate Attribute Authority to query uses the same lookup mechanisms by which the PDP was discovered. This attribute collection process is transparent to the attribute transformation process. Therefore, the transformation process treats pushed and pulled attributes identically.

2.7.3 Making the Decision

Once a request is validated, the PDP must reach an access control decision. This is done in several steps, including: passing the request context to the XACML PDP and
building the authorization decision response.

**Pass Request Context to XACML PDP**

The populated request context is enclosed in the context in a SOAP message destined for the XACML PA that maintains the policy for the desired resource. The payload of the response received contains the evaluation decision made by that XACML PA. The XACML PA locates the applicable policy via the target matching functionality within the XACML specification. The PA shields PDPs from the details of indexing, storage, and retrieval of policies. The PDP needs only understand the potential results that the XACML PA may return. These potential results are:

1. “Not Applicable” if no policy target matches the request
2. “Permit” if the evaluation of the policy rules are satisfied
3. “Deny” if the evaluation of the policy rules fail
4. “Indeterminate” if an error occurs during processing or if the Authorization Authority provided insufficient information to complete evaluation.

**Build the authorization decision response**

Once the PDP receives the PA response, it creates a SAML authorization decision statement (SAML ADS) to return to the requester. First, the Authorization Authority maps XACML result to an appropriate SAML DecisionType:

1. “Not Applicable” and “Deny” map to **Deny**
2. “Permit” maps to **Permit**
3. “Indeterminate” maps to **Indeterminate**
Errors at the XACML level map to a deny decision in the SAML ADS. The response then contains a SAML status code Responder with a secondary status code of RequestDenied. This error does not reflect a SOAP or HTTP level error since processing completed successfully.

The PDP then packages the decision with information from the initial request and potentially with ancillary attribute statements. This implies that the PDP established a session for the duration of the authorization transaction. The operations that occur within a session include evaluating a request, generating, and interpreting attribute queries, communicating with the XACML PA, building ancillary attribute statements to provide additional information to the PEP and finally generating the response. The protocol elements of SAML provide a natural basis to associate these exchanges with a session via the unique request and response identifiers. Only when a response is returned to the initial requester does the authority discard the session as all information is packaged into a unified response. There is currently no time limit enforced on completing a request sessions.

2.8 Build Attribute Assertions

The first two phases of the authorization process, receipt of a request and evaluation of a request, also apply to attribute requests. Therefore, an AA can reapply the PDP interceptor/validator-processing model to only accept a valid Attribute Query. However, the prototype AA is configured with one additional validator that determines the cardinality of the requested attribute set.
Once the AA considers an Attribute Query valid, the AA determines what attributes it must retrieve from value stores to create assertions. The AA must first determine the identity of each attribute to create. These AttributeDesignators may be provided directly in the AQ; otherwise, the AA must determine all AttributeDesignators that it supports for the specified subject. For the prototype, that information is maintained in a proprietary provisioning system data dictionary. The data dictionary contains attribute identities, attribute meta-data, and sufficient data to build value queries against the attribute data-store. For example, an AA backed by a RDBMS could use table name, schema owner, and column names to extract attribute values.

Cardea associates each unique attribute identity with one or more AttributeBuilders that abstracts the details of interaction with the provisioning system. Each attribute builder can generate AttributeStatements containing the appropriate values for a given subject. The AA also maintains an AssertionBuilder that creates SAML Assertions from those AttributeStatements and without knowledge of the provision system.

Finally, each AA maintains a private repository of configuration data to determine any conditions, such as lifetime or audience restrictions, which must be included in an assertion. This configuration also governs whether generated assertions are signed. The AssertionBuilder is also responsible for generating any Assertion-level signatures.

2.9 Enforcement Characteristics

The final step in access control is enforcement of the decision. The Cardea
architecture specifies only the functionality that compliant mechanisms must implement. A Cardea-compliant PEP must deny an access request if it cannot implement these functions.

- The PEP must be able to validate the identity of the deciding authority. If the PEP cannot validate the identity of the issuing authority, it risks enforcing a decision issued by a non-authoritative party.

- The PEP must govern access for the resource identified in the ADS.

- The PEP must understand the actions permitted by the ADS. If the PEP cannot determine the permitted actions, it cannot limit enforcements according to the decision.

- The PEP must be able to associate enforcement data with the authorization decision that initiated the enforcement. If the PEP cannot associate this information, there is no mechanism to query the PEP for the details on the enforcement of a given decision.

Often enforcement requires data beyond a simple authorization decision and a list of enumerated actions. Cardea specifies several standard attributes that can provide common types of data used as input to authorization enforcement. If an authorization authority chooses to return this information as part of the Authorization Credential, it will be in the form of ancillary Attribute Statements contained the Authorization Assertion. These standard attributes include all attributes listed defined in Appendix I. For example, a request to execute a process on a UNIX compute host may be enforced by creating a
local account on the resource to own that process. To successfully execute the process, that local account must be configured with data such as local group memberships, quotas, and charge identity. Although the PEP may automatically establish some of these characteristics, others require additional information to properly configure enforcement.

Regardless of initiation mode, the principal must ultimately remain informed of the authorization decision. Cardea provides this capability by treating the authorization request itself as a specific identity during the enforcement phase. Therefore, existing processes for gathering information about a specific identity can be reapplied to report enforcement data. Thus, a principal may request attribute information associated with an authorization request to discover details about the enforcement of a particular authorization decision. If a principal generates the initial authorization request and then forwards a credential to the PEP, the principal is responsible for extracting the response identity from the credential. If instead the PEP generated the authorization request on behalf of the principal, the PEP is responsible for providing the principal with the response identity in some unspecified manner.

Cardea stipulates that information about the enforcement of a particular authorization decision be communicated via the SAML Attribute Query process. Therefore, each PEP becomes a lightweight Attribute Authority that is capable of generating attribute assertions relative to the authorization credentials under its enforcement. If the PEP receives an Attribute query for an authorization decision that it did not enforce or for an enforcement that has expired, the response must contain zero
assertions. The PEP can make this determination by examining the subject of the Attribute Query. This specific subject conforms to the following rules:

1. The value of the NameIdentifier element contains the responseIdentifier from the authorization credential.

2. The Format URI identifier should be set to:


See Appendix E for definitions of several standards attributes that report information pertaining to the enforcement of an authorization decision.

3 Conclusions

A distributed authorization system must supply both authorization specific and basic distributed system functionality to successfully meet the requirements of distributed authorization. These requirements include:

- Providing a robust authorization decision process that spans multiple independent security domains
- Allowing participating authorities to negotiate an authorization decision online without restricting functionality or policy internal to a particular security domain
- Ensuring that all participants in a single authorization transaction can trust and interpret information that crosses administrative boundaries
- Reducing reliance on specific identity credentialing and enforcement mechanisms.
Cardea addresses these requirements by concentrating on the characteristics unique to that combined context. These characteristics are:

- Establishing a standard language for shared comprehension
- Selecting a standard protocol for consistent, reliable interpretation
- Supporting modularity & interoperability or diversity of implementation
- Encouraging portability & platform independence for flexibility and broader usability

This approach focuses on the complexities of initiating, managing, deciding, and enforcing an authorization request that involves multiple autonomous security domains.

The initial Cardea prototype successfully proves that an authorization system based on generic identities rather than specific identities meets the requirements of a distributed authorization system for a distributed environment. Scalability of the authorization system increases dramatically as a single generic identity referenced in an authorization policy can represent any number of specific identities. Furthermore, reliance on generic identities decouples responsibility for decision and enforcement of an authorization transaction. Moreover, generic identities also provide a basis for reporting authority and attribute meta-data. Finally, generic identities provide can leverage multiple identity credentials locally without modification to existing enforcement mechanisms.

Initial prototypes also confirm that implementing the selected authorization model reduces the volume of data that must be pre-configured within a domain to support a
distributed authorization transaction. Cardea defines policies in terms of generic
ingentities. Therefore, policies need not enumerate every potential combination of specific
principal and resource identities that may require an authorization decision. Further,
Cardea collects information within the context of a single transaction. Thus, participants
in a single transaction need know only *how* to collect information rather than storing
every potentially required datum.

Cardea proves that local domains can remain autonomous over local policy and
infrastructure and simultaneously participate in a global transaction. The split of decision
and enforcement permits each security domain to leverage its existing credentialing and
enforcement infrastructures. By defining standard profiles for a variety of specific
identity representations, Cardea leverages existing credentialing infrastructures rather
than requiring a specific credentialing mechanism. Existing enforcement mechanisms
adopt one of several approaches for initiating an authorization transaction. By supporting
the mainstream variants, Cardea minimizes the changes required of enforcement
mechanisms to participate in an authorization transaction.

Cardea demonstrates that an Attribute Authority can be built from existing data
stores and systems. Successfully implementing this interface proves that the selected
standards are flexible enough to represent critical attribute data. To this end, the system
implementation recognizes several standard attribute definitions, including meta-data
attributes, enforcement attributes, and authority meta-data attributes. Therefore, virtually
any combination of attributes, which are needed to provide a robust authorization
process, can be extracted from existing stores without retooling them.

Finally, Cardea’s model for distributed authorization confirms that multiple domains can participate in the context of a single transaction and reach a common semantic agreement without introducing a central authority or synchronized mechanisms to enforce the global decision. Achieving this result requires careful consideration of the following issues: appropriate communication paradigms, common information representations and semantics, suitable authorization algorithms, and flexible authority binding methods. Cardea addresses these issues by leveraging the features of several web service standards and layering their functionality to provide an end-to-end solution.

Successively layered phases prove out these capabilities. Therefore, issues common to several phases are resolved early in the design process. The selected resolutions are then leveraged to focus on successively more complex issues in later phases.

The first phase of implementation confirmed that security policy can be written and authorization decisions can be made using only generic identities. This phase also proved that a common generic identity could abstract the specific identities used in the authorization transaction. Finally, this phase demonstrated that an appropriate generic identity could be generated from the identity credential used during authentication.

The second implementation phase defined the mechanisms used to report
authority and attribute meta-data. This ability increases scalability and flexibility of the overall system by increasing the data discovered online for a given transaction. Each increase in scalability and flexibility permits integration of a wider range of specific implementations. Supporting a wide variety of implementation options increases the feasible adoption base, which translates directly into a wider variety of supported access requests.

The final implementation phase refined the authorization and implementation support for multiple messaging modes within the system. This refinement also focused on the adopted system model of trust. This model of trust enhances overall security processing, filtering capabilities, and authority discovery mechanisms. The adopted model itself also promotes scalability and flexibility by relying only on inter-domain and authority trust rather than requiring pair-wise trust relationships.

With authorizations for a single transaction distributed across multiple resources, each authorization enforcer must acknowledge the enforcement to the requester. With multiple modes of operation and with the distributed nature of authorization transactions, a requester is not necessarily guaranteed a direct hook into each involved enforcement mechanism. Therefore, the final implementation phase architected this feedback mechanism. This mechanism leverages existing Cardea processes to provide the necessary functionality, similarly to how other processes are re-engineered from existing implementations. This process also reapplyes several system concepts, such as the treatment of an authorization decision as a principal, within these processes.
The re-use of processes within the system to resolve seemingly disparate tasks was also successfully applied to several other system functions. First, locating authorities via existing lookup mechanisms proved a successful strategy. Further, authority meta-data is reported by treating meta-data as first-class attributes that can be directly queried from the authority. Finally, attribute meta-data can be included within the attribute structure by also representing the meta-data as attributes that can be included natively into the assertion structure.

Ultimately, Cardea proves that existing components and emerging standards can be leveraged in a new manner to produce native dynamic authorization support in a distributed environment. Initial prototypes prove the viability of the model and identify the critical facets of the system architecture. Thus, native distributed authorization can be achieved and a number of limitations found in current and evolving authorization systems can be resolved.

4 Future Directions/Next Steps

Although the Cardea prototype proves the viability of natively distributed authorization, there are several facets of the problem space that will benefit from additional research. These capabilities are not the core capabilities of a distributed authorization system. Rather, they are the supporting features that address the range needs of an enterprise environment. Thus, additional research must augment Cardea’s functions with these supporting features. Although these features are not core
capabilities of a distributed authorization system, they are necessary to providing a robust solution in an enterprise environment. Two of these features re-engineer proven Cardea processes. First, existing attribute structures must be revised to comply with Version 2.0 of the SAML and XACML standards. Second, the custom interface that reports the attributes required to complete a particular authorization transaction must be replaced with a standard policy profile to report them. Then, a prototype PEP must be developed to identify any PEP specific implementation concerns, such as performance tuning, that arise. The PEP re-applies processes proven by initial Cardea prototypes; therefore the PEP does not need to prove the viability of the processes themselves. Finally, policy management tools to facilitate policy file creation, administration, and discovery, such as prototyped by the PRIMA policy creator or the KPAT component of KaOS, must develop. Facilitating XACML policy file creation for security administrators will encourage the adoption of XACML policy file representation. Creating administration and discovery tools will increase the amount of information that can be discovered online for a particular authorization transaction. Ultimately, combining these supporting capabilities with Cardea’s core distributed authorization capabilities will position the Cardea solution to uniquely provide authorization in a distributed enterprise environment.
References


Terminology/Glossary

**Account**
A formal business agreement for providing regular dealings and services between a principal and provider. (24)

**Access Control**
The process of limiting access to the resources of a system only to authorized programs, processes, or other systems (in a network). Synonymous with controlled access and limited access. (21)

**Access Control List (ACL)**
The *access control list* (ACL) is a function of secure computing, used to enforce privilege separation. The list is a data structure, usually a table, containing individual users or groups’ rights to specific system objects. (2)

**Administrator**
A person who installs or maintains a system or who uses it to manage system entities, user and/or content. An administrator is typically affiliated with a particular administrative domain. (83)

**Assertion**
A security token format(24)

**Attribute**
A distinct characteristic of an entity. An entity’s attributes are said to describe it. (24)

**Authenticate**
(1) To verify the identity of a user, device, or other entity in a computer system, often as a prerequisite to allowing access to resources in a system.
(2) To verify the integrity of data that have been stored, transmitted, or otherwise exposed to possible unauthorized modification. (21)

**Authority**
A system entity that produces assertions. (24)

**Authorization**
The process of determining, by evaluating applicable access control information, whether a subject is allowed to have the specified types of access to a particular resource. Usually, authorization is in the context of authentication. (24)

**Authorization Decision**
The result of an act of authorization. The result may be negative, that is, it may indicate that the subject is not allowed any access to the resource. (24)
**Administrative Domain**
An environment or context defined by some combination of one or more administrative policies. An administrative domain may contain or define one or more security domains. An administrative domain may encompass a single site or multiple sites. Traits defining an administrative domain may evolve over time. Administrative domains may interact and enter into agreements for providing and/or consuming services across administrative domain boundaries. (24)

**Credentials**
Data that is transferred to establish a claimed principal identity. (24)

**Cryptographic Checksum**
A one-way function applied to a file to produce a unique "fingerprint" of the file for later reference. (71)

**Digital Signature**
A value computed with a cryptographic algorithm and appended to a data object in such a way that any recipient of the data can use the signature to verify the data's origin and integrity. (See: data origin authentication service, data integrity service, digitized signature, electronic signature, signer.) (71)

**Domain**
The unique context (e.g., access control parameters) in which a program is operating. (21).

**System Entity**
An active element of a computer/network system. For example an automated process, a subsystem, a person or a group of persons that incorporates a distinct set of functionality (71)

**End Point**
An end point indicates a specific location for accessing a service using a specific protocol and data format. (83)

gateway
An agent that terminates a message on an inbound interface with the intent of presenting it through an outbound interface as a new message. Unlike a proxy, a gateway receives messages as if it were the final receiver for the message. (83)

**Identity**
A representation of a principal that is mapped to a system entity that unique refers to it (24)

**In-band**
Traversing the normal data channel or flow
Loose coupling
Coupling is the dependency between interacting systems. This dependency can be decomposed into real dependency and artificial dependency:
Real dependency is the set of features or services that a system consumes from other systems. The real dependency always exists and cannot be reduced.
Artificial dependency is the set of factors that a system has to comply with in order to consume the features or services provided by other systems. Typical artificial dependency factors are language dependency, platform dependency, API dependency, etc. Artificial dependency always exists, but it or its cost can be reduced. Loose coupling describes the configuration in which artificial dependency has been reduced to the minimum. (83)

Meta-data
Data that describe the content, quality, condition and other characteristics of data. (2)

Out-of-Band
Communication in a transmission channel that is logically independent transmission channel used to communicate data

Principal
A principal is a basic entity that participates in authentication. In most cases, a principal represents a user or an instantiation of a network service on a particular host. Its principal identifier uniquely names each principal. (39)

Policy Decision Point (PDP)
An authority that makes authorization decisions for itself or for other system entities that request such decisions. A PDP is an authorization decision authority. (24)

Policy Enforcement Point (PEP)
A system entity that requests and subsequently enforces authorization decisions. (24)

Policy
A set of rules and formats, semantic and syntactic that permits entities to exchange information. (21)

Proxy
An entity authorized to act for another. An authority or power to act for another. A document giving such authority. (83)

Public Key Infrastructure
The term "public key infrastructure" (PKI) is used to describe the processes, policies and standards that govern the issuance, maintenance and revocation of the certificates, public and private keys that the encryption and signing operations require. (72)
Public Key Cryptography

Public key cryptography allows users of an insecure network, like the Internet, to exchange data with confidence that it will be neither modified nor inappropriately accessed. This is accomplished through a transformation of the data according to an asymmetrical transformation parameterized with a pair of keys. (24)

Pull
To actively request information from a system entity.

Push
To provide information to a system entity that did not actively request it.

Resource
Data contained in an information system. A service provided by a system. An item of system equipment. A facility that houses system operations and equipment. (71)

Requester
A system entity that requests services from another system entity.

Security
A collection of safeguards that ensure the confidentiality of information, protect the systems or networks used to process it and control access to them. Security typically encompasses the concepts of secrecy, confidentiality, integrity and availability. It is intended to ensure that a system resists potentially correlated attacks. (24)

Security domain
An environment or context that is defined by security models and a security architectures, including a set of resources and a set of system entities that are authorized to access the resources. One or more security domains may reside in a single administrative domain. (83)

Security Policy
The set of laws, rules and practices that regulate how an organization manages, protects and distributes sensitive information. (83)

Service
A service is an abstract resource that represents a capability of performing tasks that form a coherent functionality from the point of view of provider entities and requester entities. To be used, a service must be realized by a concrete provider agent. (83)

Service Oriented Architecture (SOA)
SOA is an architectural style whose goal is to achieve loose coupling among interacting software agents. A service is a unit of work done by a service provider to achieve desired end results for a service consumer. Both provider and consumer are roles played by software agents on behalf of their owners. (25)
**Subject**
A principal in the context of a security domain. (24)

**URI**
A compact string of characters for identifying an abstract or physical resource. (6)

**URI Reference**
A URI that may have an appended number sign (#) and fragment identifier. Fragment identifiers address particular locations or regions within the identified resource (6)

**Web-service**
A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. (83)
Appendix A
Consider the scenario illustrated below. User1 wants to access a protected service. The service Guard uses SAML to communicate with a remote authority to determine if User1 can access the service as requested. Figure 19 depicts the basic SAML exchange. In each message, the subject is user1. In the request, the message sender is the Guard. In the response, the authority plays is the sender. The PDP provides an answer to the question contained in the <SAMLRequest>; therefore it also plays the role of authority.

![Figure 19. A sample SAML Exchange](image-url)
Appendix B
Example SAML Request Context

<?xml version="1.0"?>
  <AttributeQuery Resource="turing.nas.nasa.gov">
    <Subject xmlns="urn:oasis:names:tc:SAML:1.0:assertion">
      <NameIdentifier Format urn:oasis:names:tc:SAML:1.1:nameid-format:X.509SubjectName ">CN=Rebekah Lepre,OU=Ames Research Center,O=National Aeronautics and Space Administration,O=Grid</NameIdentifier>
    </Subject>
    <saml:AttributeDesignator AttributeName="c" AttributeNamespace="urn:ipg:names:tc:DAS:1.0:attributes"/>
    <saml:AttributeDesignator AttributeName="ou" AttributeNamespace="urn:ipg:names:tc:DAS:1.0:attributes"/>
  </AttributeQuery>
</Request>
Appendix C

Attributes that carry standard semantics for Cardea processors:

1. X.500 attributes:
   a. The semantics for each of these attributes is established in RFC2256. (80) In
      particular, note that the ‘c’ attribute represents citizenship.
   b. The AttributeNamespace XML field contains the value
      urn:oasis:names:tc:SAML:2.0:attribute-namespace:uri
   c. The AttributeName field contains the attribute identifier as specified by XACML.
      These identifiers conform to the following format:
      http://www.ietf.org/rfc/rfc2256#[shortname] where shortname is the shortname
      for the X500 attribute as defined in [RFC2256].
   d. The Attribute data type reflects the data-type as defined in [RFC2256]
   e. The Attribute scope reflects the issuing AA identity.
   f. No attribute owner is specified.

2. Group membership:
   a. The semantics for this attribute indicate this subject is a member of a particularly
      named group within the current security domain. This membership may or may
      not map to group membership on a particular compute resource.
   b. The AttributeNamespace XML field contains the value
      urn:oasis:names:tc:SAML:2.0:attribute-namespace:uri
   c. The AttributeName field contains the following value
      urn:cardea:attribute:action:group
   d. The Attribute data type reflects the following value
      http://www/w3.org/2001/XMLSchema#string
   e. The Attribute scope reflects the issuing AA identity.
   f. The Attribute owner reflects the manager or administrator for that group.

3. Charge identity:
   a. The semantics for this attribute indicate this subject can incur charges to a
      particular project within the current security domain. Representing this capability
      on particular compute resource is enforcement specific.
   b. The AttributeNamespace XML field contains the value
      urn:oasis:names:tc:SAML:2.0:attribute-namespace:uri
   c. The AttributeName field contains the following value
      urn:cardea:attribute:action:charge
   d. The Attribute data type reflects the following value
      http://www/w3.org/2001/XMLSchema#string
   e. The Attribute scope reflects the issuing AA identity.
   f. The Attribute owner reflects the manager or administrator for that charge account.

4. Role:
   a. The semantics for this attribute indicate this subject can assume the specified rule
      within the current security domain. Representing this capability on particular
      compute resource is enforcement specific.
   b. The AttributeNamespace XML field contains the value
      urn:oasis:names:tc:SAML:2.0:attribute-namespace:uri
   c. The AttributeName field contains the following value
      urn:cardea:attribute:action:role
d. The Attribute data type reflects the following value http://www/w3.org/2001/XMLSchema#string

e. The Attribute scope reflects the issuing AA identity.

f. The Attribute owner reflects the manager or administrator that granted role privileges to the current subject.
Appendix C
SAMLAttribute statement profile for carrying attribute meta-data

1. The subject of the attribute statement identifies the issuing authority
2. The AttributeNamespace XML field contains the value
   urn:oasis:names:tc:SAML:2.0:attribute-namespace:uri
3. The AttributeName XML field contains one of the following values:
4. A URI formatted value for the attribute
   a. Datatype: Attribute datatypes URIs use the XACML URI representations for xsd types. Although in some cases, such as X500 attributes, the datatype may be implicitly determined based on the attribute identity, PDP processing expects an explicit representation of the data-type. To minimize redundant processing, Cardea leverages this attribute meta-data functionality to provide that mapping.
   b. Scope: Attribute scope represents the domain in which the attribute has value. The inclusion of this attribute is intended to allow a single attribute authority issue attributes valid only in a particular sub-domain. For example, the myCompany domain might implement a single attribute authority that can issue attributes for its employees, including a specialized attribute indicating that an employee is a manager. This attribute could be scoped to each department or division within myCompany so that policy issued by each department or division applies appropriately. The value of scope thus represents the sub-domain identity as a URI.
   c. Owner: Attribute ownership reflects situations in which an external identity has vouched to the attribute authority that a given principal does indeed posses the attribute. For example, to represent that a particular employee of myCompany participates in a specific project, the attribute owner should reflect the identity of the project manager where the actual attribute value represents the project itself.
Appendix D
Authority related attribute definitions

1. The default expiration time for an issued assertion. The value indicates the amount of time from the issue point for which the assertion should be honored.
   a. The AttributeNamespace XML field contains the value
      urn:oasis:names:tc:SAML:2.0:attribute-namespace:uri
   b. The AttributeName field contains the following value
      urn:cardea:metadata:2003:10:validUntil
   c. The Attribute data type reflects the following value
      http://www/w3.org/2001/XMLSchema#integer

2. Trusted authorities. The values represent the URI formatted identities of other authorities trusted by this authority. There is no requirement that this list of trusted authorities be complete an exhaustive. The authority may choose to limit the authorities it reports.
   a. The AttributeNamespace XML field contains the value
      urn:oasis:names:tc:SAML:2.0:attribute-namespace:uri
   b. The AttributeName field contains the following value
      urn:cardea:metadata:2003:10:trusted-authorities
   c. The Attribute data type reflects the following value
      http://www/w3.org/2001/XMLSchema#integer

3. Audience Restriction enforced. The value specifies whether Assertions issued by this authority will contain the AudienceRestrictionCondition.
   a. The AttributeNamespace XML field contains the value
      urn:oasis:names:tc:SAML:2.0:attribute-namespace:uri
   b. The AttributeName field contains the following value
   d. The Attribute data type reflects the following value
      http://www/w3.org/2001/XMLSchema#boolean

4. Statement types generated by the authority. The values represent the types of statements generated by this authority. Each value indicates the XML namespace and element name of the supported statement. For example, the SAML v1.1 attribute statement is represented as:
   urn:oasis:names:tc:SAML:1.0:assertion:AttributeStatement
   a. The AttributeNamespace XML field contains the value
      urn:oasis:names:tc:SAML:2.0:attribute-namespace:uri
   b. The AttributeName field contains the following value
      urn:cardea:metadata:2003:10:statement-type
   c. The Attribute data type reflects the following value
      http://www/w3.org/2001/XMLSchema#integer

5. Authority contact information. The value is a URL that may be used to direct the principal to the authority for additional information. Each authority MUST provide this meta-data.

---

11 Although authority meta-data identities currently reflect the Cardea meta-data namespace, the final component is named in accordance with the elements definitions found authority meta-data exchange proposals slated for inclusion in SAML 2.0 when possible.
a. The AttributeNamespace XML field contains the value
   urn:oasis:names:tc:SAML:2.0:attribute-namespace:uri
b. The AttributeName field contains the following value
c. The Attribute data type reflects the following value
   urn:oasis:names:tc:SAML:2.0:attribute-namespace:uri

6. Responsible organization for the authority. The value provides a human readable
   string that identifies the organization maintaining this authority
   a. The AttributeNamespace XML field contains the value
      urn:oasis:names:tc:SAML:2.0:attribute-namespace:uri
   b. The AttributeName field contains the following value
      urn:cardea:metadata:2003:10:OrganizationDisplayName
   c. The Attribute data type reflects the following value
      http://www/w3.org/2001/XMLSchema#string

7. SAML protocols supported. The value describes the set of SAML protocols the
   authority MUST support as NMToken. Known tokens are:
   urn:oasis:tc:SAML:1.0:protocol, urn:oasis:tc:SAML:1.1:protocol,
   urn:oasis:tc:SAML:2.0:protocol
   a. The AttributeNamespace XML field contains the value
      urn:oasis:names:tc:SAML:2.0:attribute-namespace:uri
   b. The AttributeName field contains the following value
   c. The Attribute data type reflects the following value
      http://www/w3.org/2001/XMLSchema#nmToken
Appendix E
Attribute definitions pertaining to the enforcement of an authorization decision

**Enforcement Status:**

a. The semantics for this attribute indicate the current status of the enforcement.
c. The AttributeName field contains the following value `urn:cardea:metadata:2003:10:enforcement:status`
d. The Attribute data type reflects the following value `http://www/w3.org/2001/XMLSchema#string`
e. The Attribute owner reflects the identity of the PEP managing the enforcement.
f. The legal values for this attribute must include:
   i. active
   ii. inactive
   iii. pending
   iv. error

**Local Account Identifier:**

g. The value for this attribute represents a local uid used to enforce the authorization decision
h. The AttributeNamespace XML field contains the value `urn:oasis:names:tc:SAML:2.0:attribute-namespace:uri`
i. The AttributeName field contains the following value `urn:cardea:metadata:2003:10:enforcement:uid`
j. The Attribute data type reflects the following value `http://www/w3.org/2001/XMLSchema#string`
k. The Attribute owner reflects the identity of the PEP managing the enforcement.

**Local Account Name:**

l. The value for this attribute represents a local username used to enforce the authorization decision
m. The AttributeNamespace XML field contains the value `urn:oasis:names:tc:SAML:2.0:attribute-namespace:uri`
n. The AttributeName field contains the following value `urn:cardea:metadata:2003:10:enforcement:uname`
o. The Attribute data type reflects the following value `http://www/w3.org/2001/XMLSchema#string`
p. The Attribute owner reflects the identity of the PEP managing the enforcement.

**Local Group Memberships:**

q. The value(s) for this attribute represents any group locally assigned to support enforcement
r. The AttributeNamespace XML field contains the value `urn:oasis:names:tc:SAML:2.0:attribute-namespace:uri`
s. The AttributeName field contains the following value
   urn:cardea:attribute:action:group

t. The Attribute data type reflects the following value
   http://www/w3.org/2001/XMLSchema#string

u. The Attribute owner reflects the identity of the PEP managing the enforcement.

**Local Charge Identity:**

v. The value(s) for this attribute represents any group locally assigned to support enforcement

w. The AttributeNamespace XML field contains the value
   urn:oasis:names:tc:SAML:2.0:attribute-namespace:uri

x. The AttributeName field contains the following value
   urn:cardea:attribute:action:charge

y. The Attribute data type reflects the following value
   http://www/w3.org/2001/XMLSchema#string

z. The Attribute owner reflects the identity of the PEP managing the enforcement.

**Time of Enforcement:**

aa. The time that enforcement began for the authorization decision

bb. The AttributeNamespace XML field contains the value
   urn:oasis:names:tc:SAML:2.0:attribute-namespace:uri

cc. The AttributeName field contains the following value

dd. The Attribute data type reflects the following value
   http://www.w3.org/2001/XMLSchema#dateTime

e. The Attribute owner reflects the identity of the PEP managing the enforcement.

**Time Remaining for Enforcement:**

ff. The time that enforcement began for the authorization decision

gg. The AttributeNamespace XML field contains the value
   urn:oasis:names:tc:SAML:2.0:attribute-namespace:uri

hh. The AttributeName field contains the following value

ii. The Attribute data type reflects the following value
    http://www.w3.org/TR/2002/WD-zquery-operators-20020816#dayTimeDuration

The Attribute owner reflects the identity of the PEP managing the enforcement.
Appendix F
Sample Policy File

This rule verifies that all users wishing to perform any action satisfies the characteristic, using LDAP naming/format of attributes, http://www.ietf.org/rfc/rfc2256.txt#c is 'US'.

<?xml version="1.0" encoding="UTF-8"?>
<Policy PolicyId="TuringPolicy" RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable">
  <!--
  Rule to see if we should allow the Subject to login -->
  <Rule Effect="Permit" RuleId="USCitizenRule">
    <Description>This rule verifies that all users wishing to perform any account management (Map, Unmap, Query) on Turing satisfies the characteristics, using LDAP naming/format of attributes
    1) http://www.ietf.org/rfc/rfc2256.txt#c is 'US'
    2)http://www.ietf.org/rfc/rfc2256.txt#o is 'National Aeronautics and Space Administration'</Description>
    <!-- Only use this Rule if the action is login -->
    <Target>
      <Subjects>
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:match:attribute-value-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">US</AttributeValue>
          <SubjectAttributeDesignator AttributeId="http://www.ietf.org/rfc/rfc2256.txt#c" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </SubjectMatch>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <AnyAction/>
      </Actions>
    </Target>
    <!-- Only allow logins from 9am to 5pm -->
  </Rule>
  <!-- A final, "fall-through" Rule that always Denies -->
  <Rule Effect="Deny" RuleId="FinalRule"/>
</Policy>
Appendix G
Authorization Decision Profile

1. An authorization request must be formatted as a SAML
AuthorizationDecisionQuery (ADQ).
   a. The subject of the query MUST identify the principal requesting
      authorization.
   b. The requested resource and actions MUST be stated in the query.
   c. Any attributes to be “pushed” to a PDP within an ADQ must be present as
      signed SAML Assertions within the Evidence element of the query.

2. Any assertions holding meta-data statements must be contained in the Advice
   element of an assertion enclosing the affected statements.
   a. All meta-data statements within the Advice element apply to each
      statement enclosed within the assertion.
   b. Each meta-data statement type must appear at most once in a particular
      assertion to avoid ambiguity.
The first profile relies on X.509 subject names. This representation has the following characteristics:

1. The SAMLSubject contains a NameIdentifier child element.
2. The content of the NameIdentifier element is the distinguished name from the X.509 certificate formatted according to the requirements for the ds:X.509SubjectName element in the XMLDSig recommendation.
3. The value of the Format attribute is: urn:oasis:names:tc:SAML:1.1:nameid-format:X.509SubjectName

The second profile relies directly on X.509 certificates. This representation has the following characteristics:

1. The SAMLSubject contains a SubjectConfirmation child element.
2. The content of the SubjectConfirmationData element is a base64 encoded X.509 certificate.
3. The value of the SubjectConfirmationMethod attribute is:

   urn:oasis:names:tc:SAML:1.0:am:X.509-PKI

The third profile relies XML digital signatures. This representation has the following characteristics:

1. The SAMLSubject contains a SubjectConfirmation child element.

---

12 If an authority guarantees an assertion with a short-lived certificate, the lifetime of any assertion signed with the associated key-pair must be at most the lifetime of the key pair.
2. The SubjectConfirmation element contains a ds:KeyInfo element.

3. The value of the SubjectConfirmationMethod attribute is: urn:ietf:rfc:3075

13 This encoding is the same encoding used when representing a digital certificate in a WSS construct.
Appendix I
Authorization Decision Profile

- Attribute identity is comprised of two pieces, an attribute classification scheme and a unique and unambiguous attribute name scoped to that classification scheme.

- Data-type is a mandatory meta-data attribute. However, it is not considered part of the attribute identity.

- Any other meta-data attribute is optional.

- When requesting attributes from an authority, a requester provides the classification scheme and name (identity) of any desired attribute for a specific identity.

- A response packages attribute identity, data-type, appropriate values and potentially other meta-data into a SAMLAttributeStatement.
Appendix J
XACML Profile

1. The attribute name must be a URI so that it may map to the XACML attributeid.

2. The value of the Issuer attribute of the SAMLAssertion carrying the attribute statement becomes value of Issuer of the generated XACML attribute.

3. The value of the attribute data-type meta-data statement must be a URI. This value maps to the DataType of the XACML attribute.

4. According to XACML 1.1, each Attribute Statement may carry at most a single Attribute value. Therefore, each AttributeValue contained in a SAML attribute statement generates a separate XACML attribute each of which carries identical attribute meta-data.
Appendix K
Authorization Decision Rules

a. The requester cannot mandate SOAP level headers be included in the response.
b. The requester must authenticate the identity of the subject before generating the AuthorizationDecisionQuery.
c. The subject must be formatted as a SAML Subject (see section 2.5.2)
d. The requested resource must be identified by a URI
e. Any attribute assertions pushed to the PDP must be included in the Evidence element of the AuthorizationDecisionQuery
f. The AuthorizationDecisionQuery must include each action for which authorization is requested. Cardea recognizes the following SAML defined action namespace identifiers and their relevant values:
   a. urn:oasis:names:tc:SAML:1.0:action:rwedc
   b. urn:oasis:names:tc:SAML:1.0:action:unix

g. Cardea recognizes the following action namespace:
   a. urn:cardea:names:action:authorize

h. Cardea supports the following values for actions within that namespace:
   1. connect – The subject of the AuthorizationDecisionQuery should be permitted to connect to the resource.
   2. disconnect – The subject of the AuthorizationDecisionQuery should be permitted to disconnect from the resource.
   3. touch – The subject of the AuthorizationDecisionQuery should be permitted to “touch” the current resource connection.
i. The SAML request construct must contain all the required attributes and
   elements identified in the SAML specification. However, the optional
   `RespondWith` element may be omitted.

j. The SAML request construct need not be digitally signed directly.
Appendix L
Authorization Decision Rules

a. If the Security Header is not present in the request, the XML firewall does not forward the request to the PDP and returns the wsse:FailedAuthentication fault code.

b. If an error occurred while processing the Security Header, the XML firewall does not forward the request to the PDP and returns the wsse:InvalidSecurity fault code.

c. If the signature was deemed invalid, the XML firewall does not forward the request to the PDP and returns the wsse:FailedCheck fault code.

d. If the Security Header processed correctly and the signature was syntactically valid but the requester failed the filter checks, the XML firewall does not forward the request to the PDP and returns the wsse:InvalidSecurityToken fault code.

e. The SOAP fault should reflect the env:Sender SOAP fault.

f. For all processing failures that occur during WSS processing, the http status should be “500 Internal Server Error”. The response should contain a SOAP message with the appropriate SOAP fault element.

If a request passes all WSS processing but fails at the SAML level, the SAML response payload will contain only a Status construct. As this failure is application specific, it will not be detectable at the SOAP or http level.