OSCARS Policy Interface

1 Goals

This document introduces the protocol used by the OSCARS Interdomain Controller (IDC) software to request policy decisions during circuit reservation. It provides use cases, an example implementation design, and a web service message definition.

The ideas in this document are intended to be extensible to other use cases but describing those use cases is beyond the scope of this document. The protocol is also neutral as to how policy decisions are made or what policies are defined. A detailed description of the policy decision process and policy definition is outside the scope of this document.

The protocol addresses many of the policy issues observed in OSCARS but may be extended or replaced by ongoing work in groups such as DICE. The end of this document highlights some of the current limitations of the protocol as currently defined.

2 Use Cases

2.1 Single Interdomain Controller

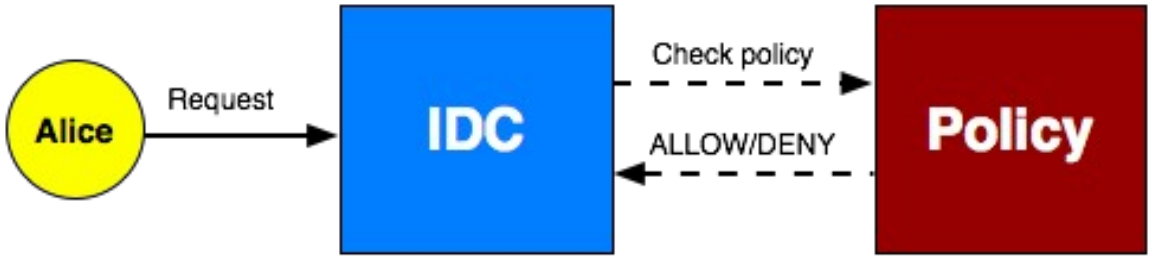

Figure 1: Simple use case where requester sends message to IDC and IDC asks policy engine if the action is allowed.

Figure 1 shows a basic use case of the policy protocol defined in this document. In the picture there are 3 entities:

1. **Alice** – the requester asking to reserve a set of network resources
2. **IDC** - the service that reserves the network resources based on the request from Alice. If Alice's request violates any policy then the IDC will not provision the network resources and return a failure.
3. **Policy Service** - the service that accepts the details of a request from the IDC and returns whether the request is allowed according to policy.

In the example the IDC asks the policy service to make a decision based on three categories of information:

1. *Who made the request*. In this case Alice made the request so we call Alice a **Requester**.

2. *The type of request.* In the context we only care about two request types: createReservation and modifyReservation[1]. We call the request type the **Action.**
3. *The details of the reservation* such as path, bandwidth and duration. We call the reservation the **Resource**.

In this example those three pieces of information are all that the IDC directly knows about a request. As we'll see in the multi-domain case this changes - especially as it pertains to the Requester.

2.2 Multiple Interdomain Controllers

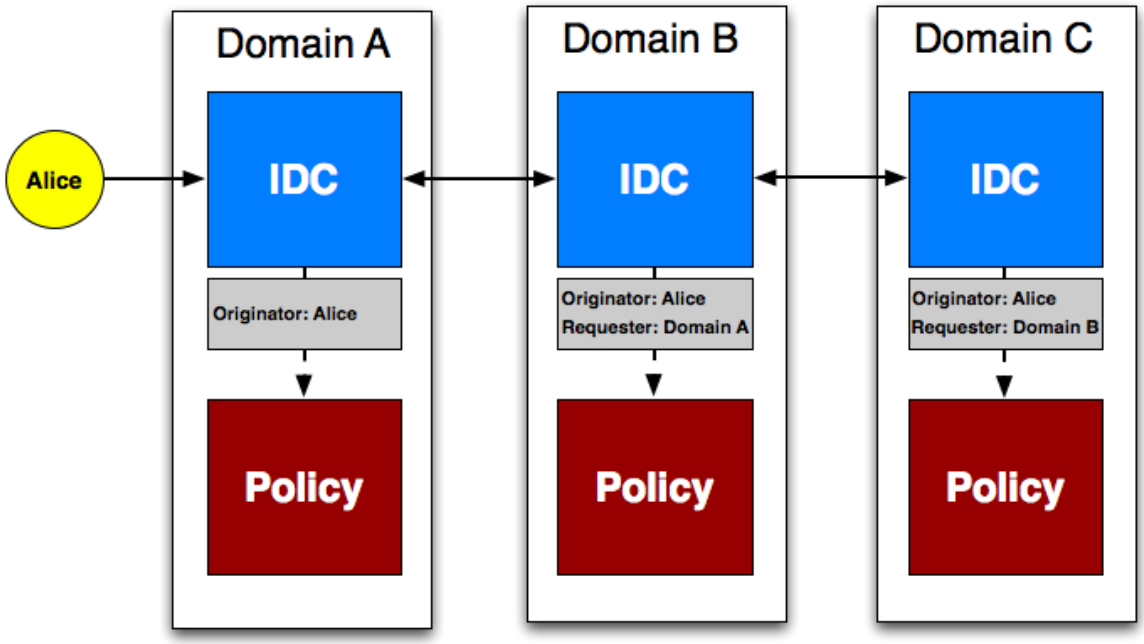

Figure 2 Multi-IDC use case where there are three domains each with their own policy service. Each domain wants policy applied based on the original requester (Alice) and the neighboring domain that forwarded it the request.

Figure 2 shows the case where there are multiple IDCs each in a different domain. Each domain has its own policy service in the example. In practice, some domains may share a policy service and others may not have any, but neither case significantly changes the main challenges of multi-domain reservation.

In the multi-domain case an initial requester sends a message to the first domain in the path. In Figure 2 Alice is the initial requester and Domain A is the first domain. If it is determined the path of the circuit to be reserved goes across multiple domains then the first IDC forwards it to the next domain in the path. In

[1] Policy applies to modifyReservation because a user could potentially use it to increase the amount of time resources are reserved. Other requests such as cancel release the resources so we are assuming that using less is always allowed.

this case Domain A forwards to Domain B. If the next domain determines there is another domain in the path the request is again forwarded. This process continues until the last domain in the path is reached. In our example Domain B forwards to Domain C and the chain ends.

A challenge arises because groups would like to be able to apply policy based on not only who sent the request (the **requester)** to their IDC but also who originally sent the request (a special type of **requester** referred to as the **originator**). The originator is an additional piece of information not required in the single domain case. Figure 2 shows the last requester and originator of each policy access request. ***Current IDC policy implementations only concern themselves with the originator and last requester.*** Future implementations and/or other use cases may also care about the intermediate requesters. The protocol allows for additional assertions about intermediate requesters to be specified but does not require it.

3 Example Policy Service Designs

3.1 Internet2 DCN Pilot

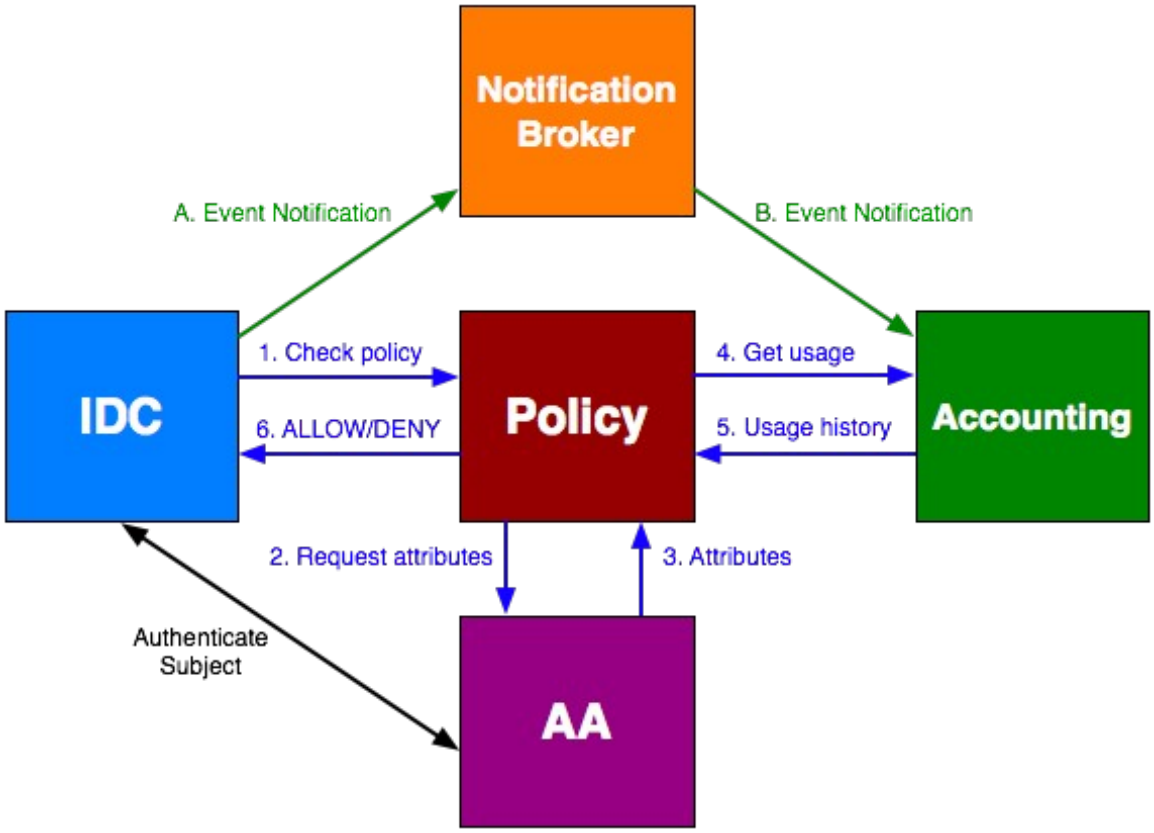

Figure 3 An example policy service design where additional information about a subject and/or resource can be polled from an external Authentication and Authorization (AA) service and Accounting service.

Figure 3 shows an example design of a policy service for the Internet2 DCN Pilot

service[2]. There are many ways a policy service could be implemented but this one is designed to meet the needs defined by the Internet2 community. This particular policy service works as follows:

- A request comes into the IDC and the IDC then authenticates that user with an Authentication and Authorization (AA) service.
- Events happen as the IDC schedules the reservation and result in the following:
    A. A notification of the event is sent to the NotificationBroker for distribution
    B. The notification is forwarded to the Accounting service and recorded
- The IDC needs policy decisions when a create or modify request is received, after a path calculation (that may just contain the next domain) and after a path is confirmed (has the full path). This results in the following process:
    1. The IDC sends a check policy request to the policy service
    2. The policy service request attributes about the subject and/or originator (if applicable) from the AA service.
    3. The AA service returns any attributes it has
    4. The policy service requests usage information for the subject and/or originator from the accounting service.
    5. The accounting service returns the requested usage information
    6. The policy service compiles the information and determines if the action should be allowed or denied. It returns the result to the IDC. The IDC then enforces the decision.

This design assumes that the IDC only provides the policy service with an identifier for the user that can be used to lookup any further information it needs. In the future it may be possible to push information such as attributes but that is not required in this design.

4 Messages and Types

4.1 checkPolicyRequest

The *checkPolicyRequest* message is sent from an IDC to the Policy service when an IDC would like to ask for a policy decision. The *checkPolicyRequest* format is described below:

```
<xsd:complexType name="checkPolicyRequestType">
    <xsd:sequence>
        <xsd:element name="Requesters"
              type="xsd:RequesterListType" />
        <xsd:element name="Action" type="xsd:anyURI" />
        <xsd:element name="Resource"
              type="tns:resourceType" minOccurs="1"
              maxOccurs="unbounded" />
```

[2] https://spaces.internet2.edu/display/DCN/DCN+Pilot

```
    </xsd:sequence>
</xsd:complexType>
```

Requesters

An ordered list of entities that have asked for the specified *Action* to be performed on the given *Resource*(s). The list MAY represent an ordered sequence or requesters in a chain. If it does represent a chain then the first requester in the sequence is the originator of the action request. The last item in the sequence represents entity that last forwarded the action request before reaching the entity that sent the checkPolicyRequest. The list SHOULD AT LEAST contain the originator and the requester that last forwarded the action request. It MAY contain intermediate requesters.

Action

Uniform Resource Identifier (URI) representing the action that the subject wished to perform. Valid URIs for the IDC case are defined in section **4.1.5.**

Resource

One or more resources on which the requesters want to perform specified action. In the case of the IDC this is the reservation. See section **4.1.6** for more information.

4.1.1 Requesters

The *Requesters* element lists the entities that requested a specified action on some given resource(s).

```
<xsd:complexType name="RequesterListType">
        <xsd:sequence>
            <xsd:element name="Requester"
                type="tns:RequesterType"
                minOccurs="1"
                maxOccurs="unbounded" />
        </xsd:sequence>
    </xsd:complexType>
```

Requester

A list of one or more Requester elements. It MAY represent a an ordered sequence of requesters in a chain.

4.1.2 Requester

A requester element describes an entity that requested a specified action on the given resource(s). This element also identifies the entity's position in a request chain if such a chain exists. This element is show below:

```
<xsd:complexType name="RequesterType">
    <xsd:sequence>
        <xsd:element name="Subject" ref="saml:Subject" />
        <xsd:element name="SubjectAuthentication"
            type="tns:SubjectAuthenticationType"
            minOccurs="0"/>
        <xsd:element ref="saml:Assertion" minOccurs="0"
```

```
            maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="sequenceNumber"
            type="xsd:integer" use="optional"/>
</xsd:complexType>
```

Subject
: A required SAML 2.0 subject that represents an entity that sent the original action request or that forwarded the request at some point in a chain. See section **4.1.3** for more information.

SubjectAuthentication
: An optional element that contains a single *saml:NameID* pointing to where more information about the Subject can be found. See section **4.1.4** for more information.

Assertion[3]
: Zero or more SAML 2.0 assertions about the *Subject* specified by this *Requester* element. See [SAML] for a description.

@sequenceNumber
: An optional integer attribute that indicates the requester's location in a forwarding chain. If this field is specified, then the originator MUST have a sequenceNumber set to 1. All subsequent *Requester* elements in the chain MUST set their *sequenceNumber* in increasing order with respect to their location in the request chain as it relates to the originator. This field SHOULD be set when a chain model is used for requests and MAY be set for non-chain models (i.e. a tree model).

4.1.3 Subject

The *Subject* element is defined in SAML 2.0. Its type definition and a description of how implementations will use each element are provided below:

```
<complexType name="SubjectType">
    <choice>
        <sequence>
            <choice>
                <element ref="saml:BaseID"/>
                <element ref="saml:NameID"/>
                <element ref="saml:EncryptedID"/>
            </choice>
            <element ref="saml:SubjectConfirmation"
                 minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
        <element ref="saml:SubjectConfirmation"
             maxOccurs="unbounded"/>
    </choice>
```

[3] The *Assertion* element will not be used in the first version of the Policy Service implemented for the Internet DCN pilot service

```
</complexType>
```

BaseID, NameID, EncryptedID

Only **NameID** will be used by near-term implementations of the IDC use case. The value of name ID will either be the X.509 subject or the OSCARS user login (if no X.509 subject is available)

NameID@Format

Either **urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName** (if an X.509 Subject) or **urn:dcn:oscars:loginName** (if an OSCARS login name)

SubjectConfirmation

The Method attribute is always set to **urn:oasis:names:tc:SAML:2.0:cm:sender-vouches**. Future implementations may want to explore different ways to use this field.

4.1.4 SubjectAuthentication

In a *Requester* element the *SubjectAuthentication* type defines a location where information about a Subject can be verified and/or found. Its definition is shown below:

```
<xsd:complexType name="SubjectAuthenticationType">
     <xsd:sequence>
        <xsd:element ref="saml:NameID" />
     </xsd:sequence>
  </xsd:complexType>
```

NameID

A URL where information about the subject can be verified and/or more information can be found. The URL may point to an identity provider, the original service that authenticated the request, or some other service.

4.1.5 Action

Currently two URIs are identified for the IDC use case. One is for creating a new reservation and the other for modifying an existing. They are as follows:

- urn:dcn:oscars:action:createReservation
- urn:dcn;oscars.action:modifyReservation

4.1.6 Resource

A resource has the following type definition:

```
<xsd:complexType name="resourceType">
    <xsd:sequence>
        <xsd:any namespace="##other" processContents="lax"
            minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
```

Any Type

One or more XML elements that describe the resource. In the IDC context it is a *reservationResource* element of type *createResContent*. It contains all the information of a request such as path, bandwidth and duration.

4.2 checkPolicyResponse

The checkPolicyResponse is returned by the Policy Service to the IDC and indicates whether the subject is allowed to perform the action on the specified resource(s). The type definition is below:

```
<xsd:complexType name="checkPolicyResponseType">
    <xsd:sequence>
         <xsd:element name="allow" type="xsd:boolean" />
         <xsd:element name="reason" type="xsd:string"
              minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
```

allow

A Boolean that's true if allowed and false if denied.

reason

An optional string describing why a particular decision was made.

4.3 Examples

4.3.1 Alice to Domain A

The following would be in the body of a SOAP request:

```
<policy:checkPolicyRequest
    xmlns:policy="http://oscars.es.net/OSCARS/policy
    xmlns:idc="http://oscars.es.net/OSCARS"
    xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
    <policy:Requesters>
        <policy:Requester sequenceNumber="1">
            <saml:Subject>
                <saml:NameId
Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:X509SubjectName">
             CN=Alice, OU=OASIS Interop Test Cert, O=OASIS
               </saml:NameId>
               <saml:SubjectConfirmation
Method="urn:oasis:names:tc:SAML:2.0:cm:sender-vouches" />
            </saml:Subject>
            <policy:SubjectAuthentication>
                 https://domainA.net/AuthN
            </policy:SubjectAuthentication>
        <policy:Requester>
    </saml:Requesters>

    <policy:Action>
         urn:dcn:oscars:action:createReservation
    </policy:Action>

    <policy:Resource>
```

```
        <idc:reservationResource>
            <idc:startTime>...</idc:startTime>
            <idc:endTime>...</idc:endTime>
            <idc:bandwidth>...</idc:bandwidth>
            <idc:description>...</idc:description>
            <idc:pathInfo>
                <idc:pathSetupMode>...</idc:pathSetupMode>
                <idc:layer2Info>
                  <idc:srcEndpoint>...</idc:srcEndpoint>
                  <idc:destEndpoint>...</idc:destEndpoint>
                </idc:layer2Info>
            </idc:pathInfo>
        </idc:reservationResource>
    </policy:Resource>
</policy:checkPolicyRequest>
```

4.3.2 Domain A to Domain B with Alice as Originator

The following would be in the body of a SOAP request:

```
<policy:checkPolicyRequest
    xmlns:policy="http://oscars.es.net/OSCARS/policyReq"
    xmlns:idc="http://oscars.es.net/OSCARS"
    xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
    <saml:Requesters>
        <policy:Requester sequenceNumber="1">
            <saml:Subject>
                <saml:NameId
Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:X509SubjectName">
             CN=Alice, OU=OASIS Interop Test Cert, O=OASIS
              </saml:NameId>
              <saml:SubjectConfirmation
Method="urn:oasis:names:tc:SAML:2.0:cm:sender-vouches" />
            </saml:Subject>
            <policy:SubjectAuthentication>
                 https://domainA.net/AuthN
            </policy:SubjectAuthentication>
        <policy:Requester>

        <policy:Requester sequenceNumber="2">
            <saml:Subject>
                <saml:NameId
Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:X509SubjectName">
         CN=DomainA, OU=DCS Test, O=DCS Test, ST=MI, C=US
                </saml:NameId>
                <saml:SubjectConfirmation
Method="urn:oasis:names:tc:SAML:2.0:cm:sender-vouches" />
            </saml:Subject>
            <policy:SubjectAuthentication>
                 https://domainB.net/AuthN
            </policy:SubjectAuthentication>
```

```
        </policy:Requester>
    </policy:Requesters>

    <policy:Action>
         urn:dcn:oscars:action:createReservation
    </policy:Action>

    <policy:Resource>
         <idc:reservationResource>
             <idc:globalReservationId>
                 ...
            </idc:globalReservationId>
            <idc:startTime>...</idc:startTime>
            <idc:endTime>...</idc:endTime>
            <idc:bandwidth>...</idc:bandwidth>
            <idc:description>...</idc:description>
            <idc:pathInfo>
                <idc:pathSetupMode>...</idc:pathSetupMode>
                <idc:layer2Info>
                   <idc:srcEndpoint>...</idc:srcEndpoint>
                   <idc:destEndpoint>...</idc:destEndpoint>
                </idc:layer2Info>
              <idc:path>....</idc:path>
           </idc:pathInfo>
       </idc:reservationResource>
    </policy:Resource>
</policy:checkPolicyRequest>
```

4.3.3 Alice (OSCARS Login) to Domain A

The following would be in the body of a SOAP request:

```
<policy:checkPolicyRequest
    xmlns:policy="http://oscars.es.net/OSCARS/policy
    xmlns:idc="http://oscars.es.net/OSCARS"
    xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">

    <policy:Requesters>
        <policy:Requester sequenceNumber=”1”>
            <saml:Subject>
                <saml:NameId
                   Format="urn:dcn:oscars:login"
                   NameQualifier=”@DomainA”>
                       alice
               </saml:NameId>
               <saml:SubjectConfirmation
Method="urn:oasis:names:tc:SAML:2.0:cm:sender-vouches" />
            </saml:Subject>
            <policy:SubjectAuthentication>
                 https://domainA.net/AuthN
            </policy:SubjectAuthentication>
        <policy:Requester>
    </saml:Requesters>
```

```
    <policy:Action>
        urn:dcn:oscars:action:createReservation
    </policy:Action>

    <policy:Resource>
        <idc:reservationResource>
            <idc:startTime>...</idc:startTime>
            <idc:endTime>...</idc:endTime>
            <idc:bandwidth>...</idc:bandwidth>
            <idc:description>...</idc:description>
            <idc:pathInfo>
                <idc:pathSetupMode>...</idc:pathSetupMode>
                <idc:layer2Info>
                  <idc:srcEndpoint>...</idc:srcEndpoint>
                  <idc:destEndpoint>...</idc:destEndpoint>
                </idc:layer2Info>
            </idc:pathInfo>
        </idc:reservationResource>
    </policy:Resource>
</policy:checkPolicyRequest>
```

5 Limitations

- The current checkPolicyResponseType is not rich enough to handle advanced use cases. One example is a case where a service wants to input a topology graph and have the output be a pruned version of the input based on policy.
- The expected implementations assume that attributes from a user can be pulled from a local AA service. Further investigation into pushing SAML assertions and/or a distributed mechanism to lookup attributes needs to be explored.
- The model described does not address all issues of transitive trust. In the case of an IDC information passes through entities with which a given IDC may have no relationship. The model discussed assumed that anything given from a neighbor, regardless of where it travelled before, would be trusted or trust could be established by an external means. In the future this assumption may not be sufficient and these relationships should be better defined.

6 References

[SAML] http://saml.xml.org/saml-specifications
[IDC] http://www.controlplane.net/