OpenDOF

The OpenDOF Project

An
Open Distributed **O**bject **F**ramework
For The
Internet of Things

Bryant Eastham

Demonstration Preparation

OpenDOF

1

Scan to download the application

2

From the application, scan to configure

For help with the demo, go to http://elc2015.opendof.org/help

Panasonic and IoT

IoT Platform Requirements

More information about these five principles can be found at http://opendof.org.

- Secure
- Interoperable
- Flexible
- Scalable
- Reliable

- Panasonic announces the formation of the OpenDOF Project.
- Java code released, C99 and C# to follow.
- All protocol specifications are open.
- Patent non-assertion on libraries *and any implementation of the specifications.*
- Work with the AllSeen Alliance on gateways.

Demonstration

Terminology

- DOF (Architecture and Specifications)
  - **D**istributed
  - **O**bject
  - **F**ramework
  - Specifications

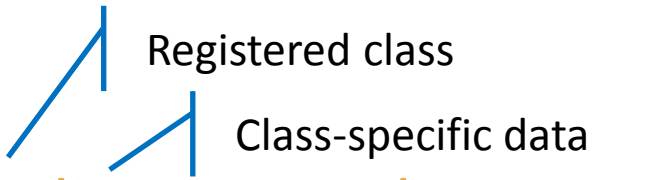
- OpenDOF (Open source implementations)

Terminology

- **Object** : distributed set of uniquely identified capabilities, bound to an Object Identifier
- **Interface** : defined set of items (properties, methods, events, exceptions) bound to an Interface Identifier
- **Identity** : unique persona associated with a secret and permissions
- **Domain** : centrally managed set of identities

Object Identifiers (OID)

- Globally unique, no registration required
- Standard text representation



[3:bryant.eastham@us.panasonic.com]

[2:{d0 67 e5 43 f8 ff}]

Interface Identifiers (IID)

- Globally unique through registration
- Standard text representation

[2:{01 07}]

Item Identifiers (ItemID)

- Unique within a single interface
- Represents an item type and data type
  - Property, Method, Event, Exception
- Defines syntax (wire format)
- Includes semantic meaning
  - Not all booleans are the same

Putting It All Together

- **Bindings** are OID plus IID
- Operations require binding and ItemID
- Context allows a short alias for the binding

Putting It All Together

Item 1 of the status interface of my computer

1 [1:{01}] [2:{d0 67 e5 43 f8 ff}]

01 05 01 02 06 d0 67 e5 43 f8 ff

Item 1 of the status interface of my computer, previously assigned alias 8

1 [1:{01}] [2:{d0 67 e5 43 f8 ff}]

01 08

Security Model

- Domains contain all security information
  - Identities (users, devices)
  - Secrets (keys, passwords)
  - Permissions

Security Model

- Each interaction typically requires two permissions
  - Permission for the request
  - Permission for the response

Security Model

- Identities are granted permissions
  - As requestors
  - As providers
  - As both requestors and providers (bridge or gateway)

API Introduction

- High-level API
  - Hides much of the lower level protocol detail
  - Removes fine-grained control over packets
- Written for the most general case
  - Not always the most scalable
- APIs are hard – they never please everyone

Example – Instantiate A DOF

```
import org.opendof.core.oal.*;
DOF.Config dofConfig;
DOFSystem.Config sysConfig;
DOFCredentials user;

user = DOFCredentials.Password.create(
       DOFObjectID.Domain.create( "[6:bar.com]" ),
       DOFObjectID.Identity.create( "[3:foo@bar.com]" ),
       "password" );
dofConfig = new DOF.Config.Builder().build();
sysConfig = new DOFSystem.Config.Builder()
                .setCredentials( user ).build();

DOF dof = new DOF( dofConfig );
DOFSystem system1 = dof.createSystem( sysConfig );
DOFSystem system2 = dof.createSystem( sysConfig );
```

opaque container used throughout the API – hides secret, represents user

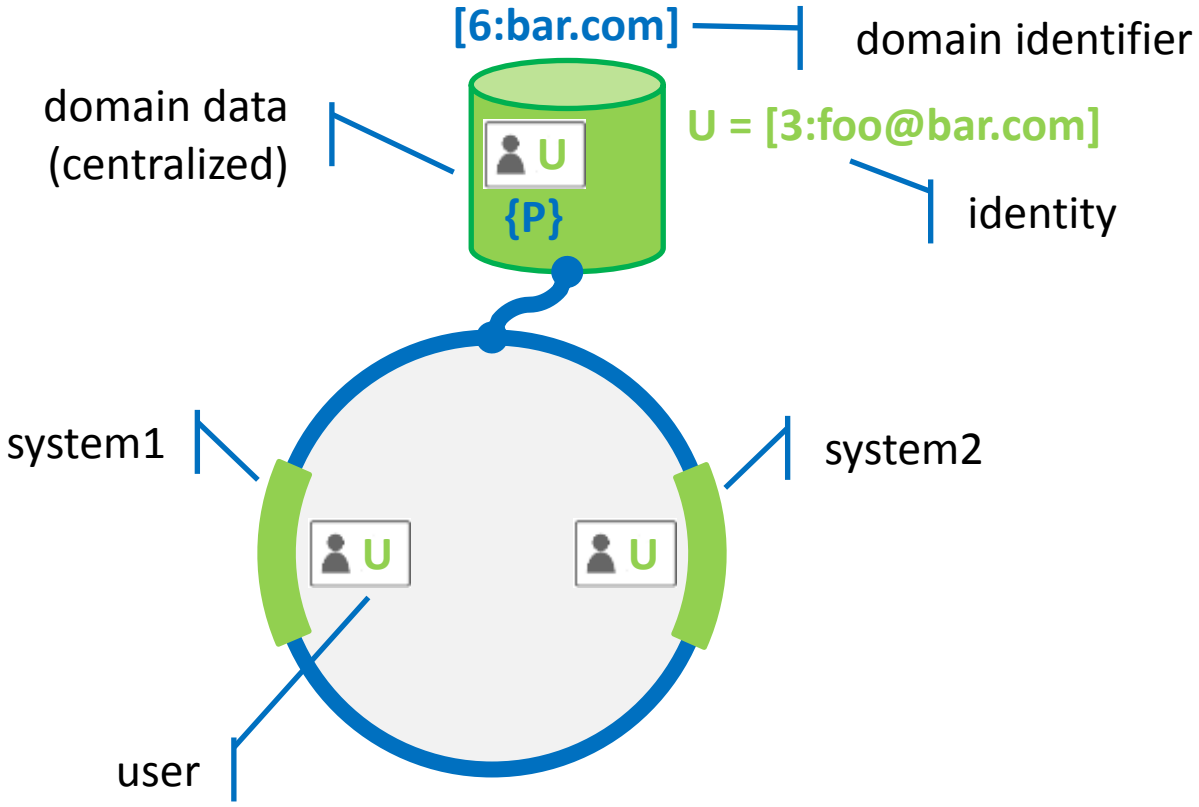
domain

secret

identity

application interacts with the DOF

Result



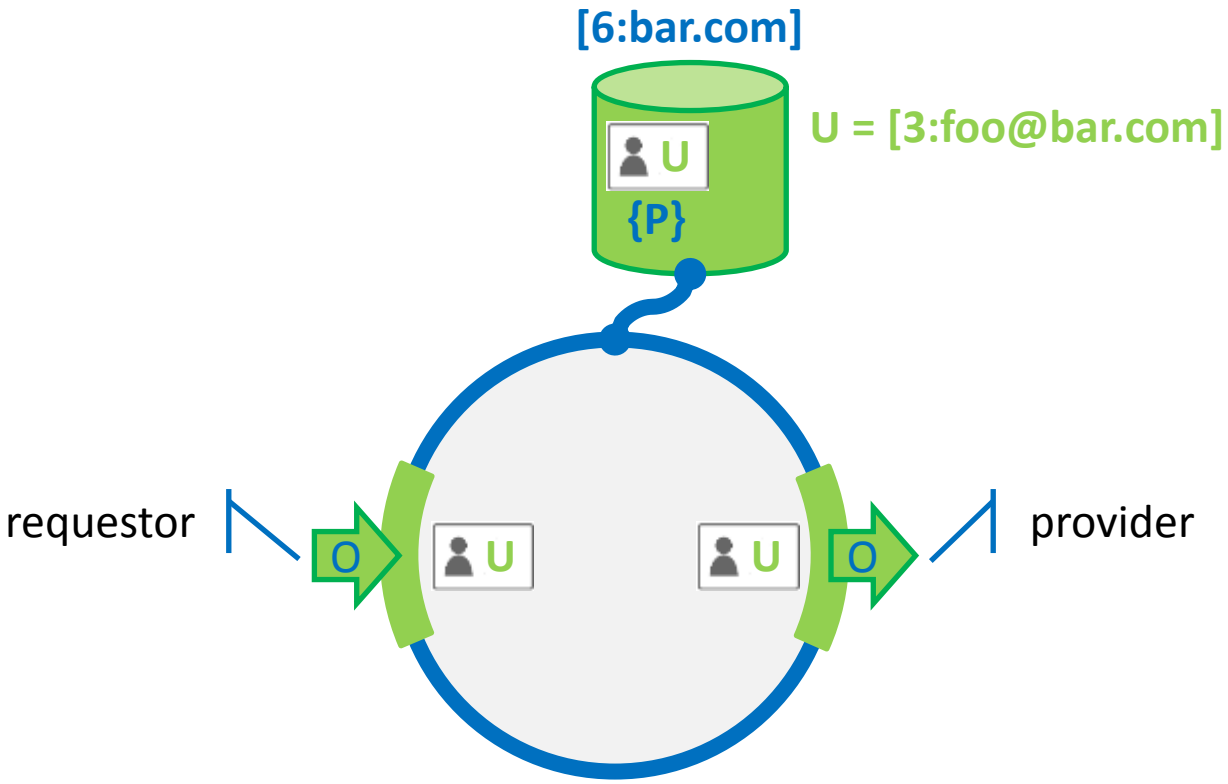
Example – Instantiate An Object

```
import org.opendof.core.oal.*;
DOFObjectID oid;
DOFObject requestor, provider;

oid = DOFObjectID.create( "[2:{d0 67 e5 43 f8 ff}]" );
requestor = system1.createObject( oid );
provider = system2.createObject( oid );
```

OID = unique identifier

objects

Result



Example – Provide An Interface

```
import org.opendof.core.oal.*;
DOFOperation provide;

provide = provider.beginProvide( Status.DEF,
                new ProvideListener() );

private class ProvideListener extends
                        DOFObject.DefaultProvider {
        public void get( Provide op,
                                DOFRequest.Get request,
                                Property property ) {
                request.respond( new DOFUInt8( 0 ) );
        }
}
```

includes IID as well as definition

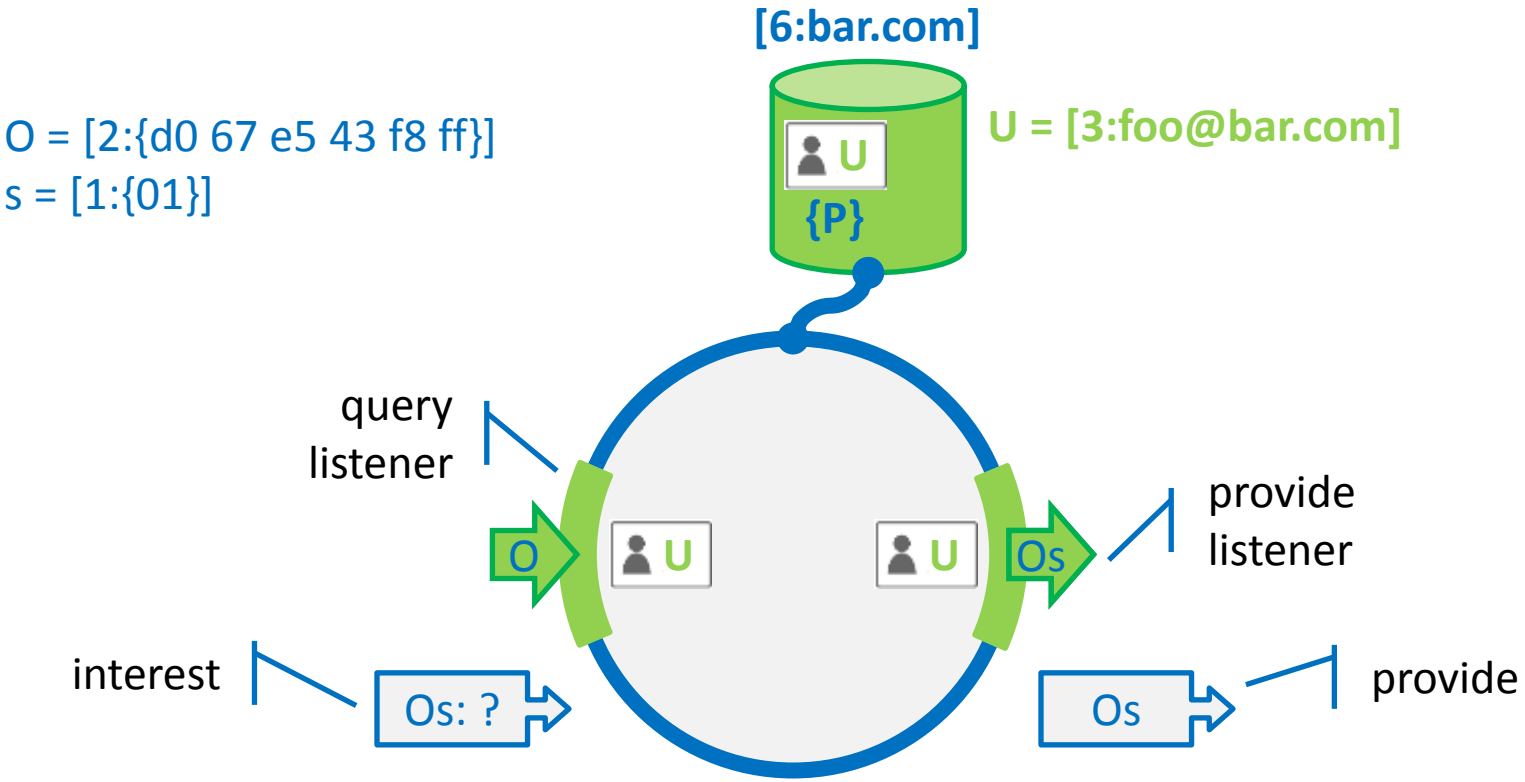
Example – Discover A Provider

```
using org.opendof.core.oal;
DOFOperation interest;
DOFQuery query;

interest = system1.beginInterest( oid, Status.IID,
                DOFInterestLevel.WATCH );
query = new DOFQuery.Builder()
                .addFilter( oid, Status.IID )
                .build();
system1.beginQuery( query, timeout, new QueryListener(), null );

class QueryListener :
                DOFSystem.QueryOperationListener {
        public void interfaceAdded( query, oid, iid ) …
        public void interfaceRemoved( query, oid, iid ) …
        public void providerRemoved( query, oid ) …
}
```

Network request

Local request

Result



Example – Get From Provider

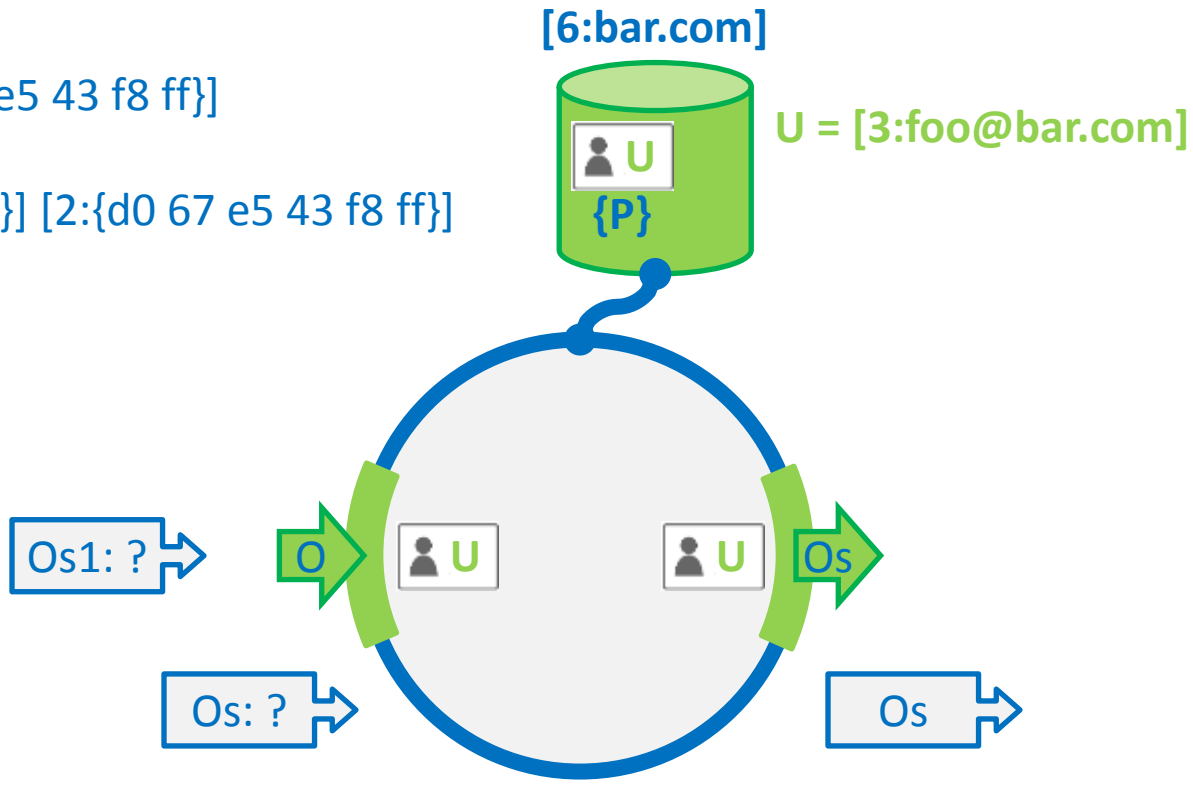
```
import org.opendof.core.oal.*;
DOFResult<DOFValue> result;
int timeout = 5000;

result = requestor.get( Status.VALUE, timeout );
int value = DOFType.asInt( result );
```

Interactions include

- Session (end-to-end tunnel)
- Property get/set/subscribe
- Method invoke
- Event register

Result

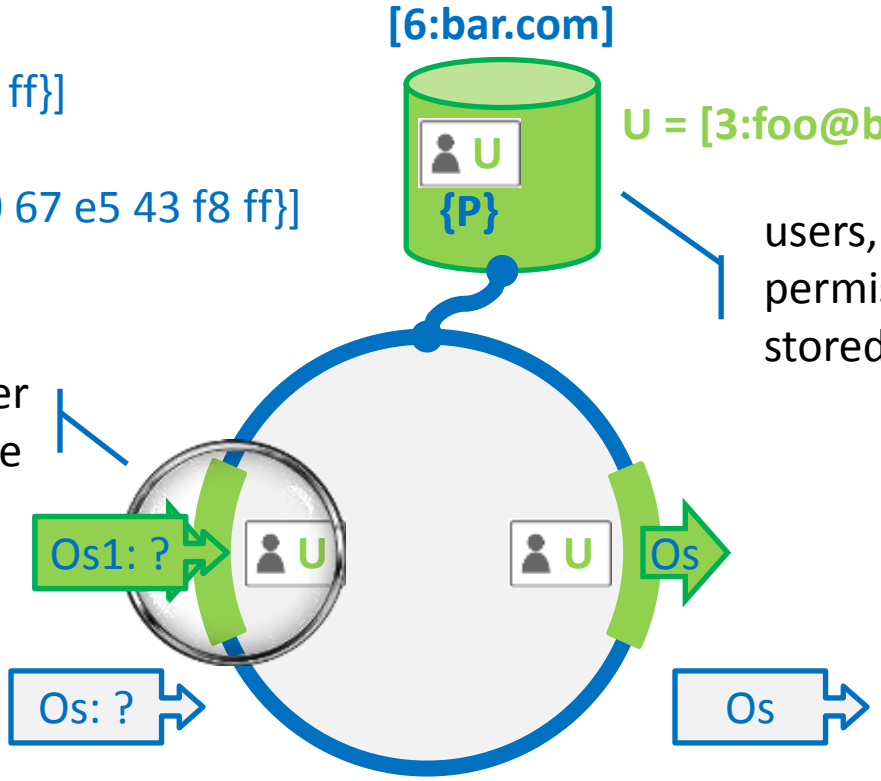
O = [2:{d0 67 e5 43 f8 ff}]
s = [1:{01}]
Os1 = 1 [1:{01}] [2:{d0 67 e5 43 f8 ff}]



Result

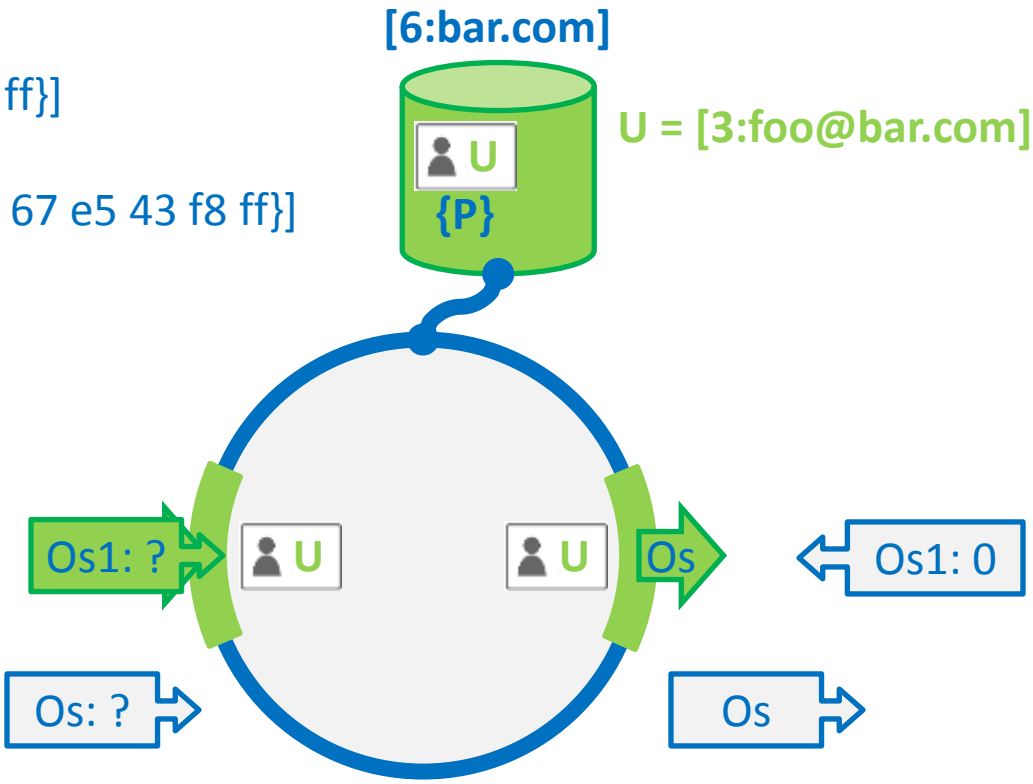
O = [2:{d0 67 e5 43 f8 ff}]
s = [1:{01}]
Os1 = 1 [1:{01}] [2:{d0 67 e5 43 f8 ff}]

all interactions are validated based on user and permissions before being accepted



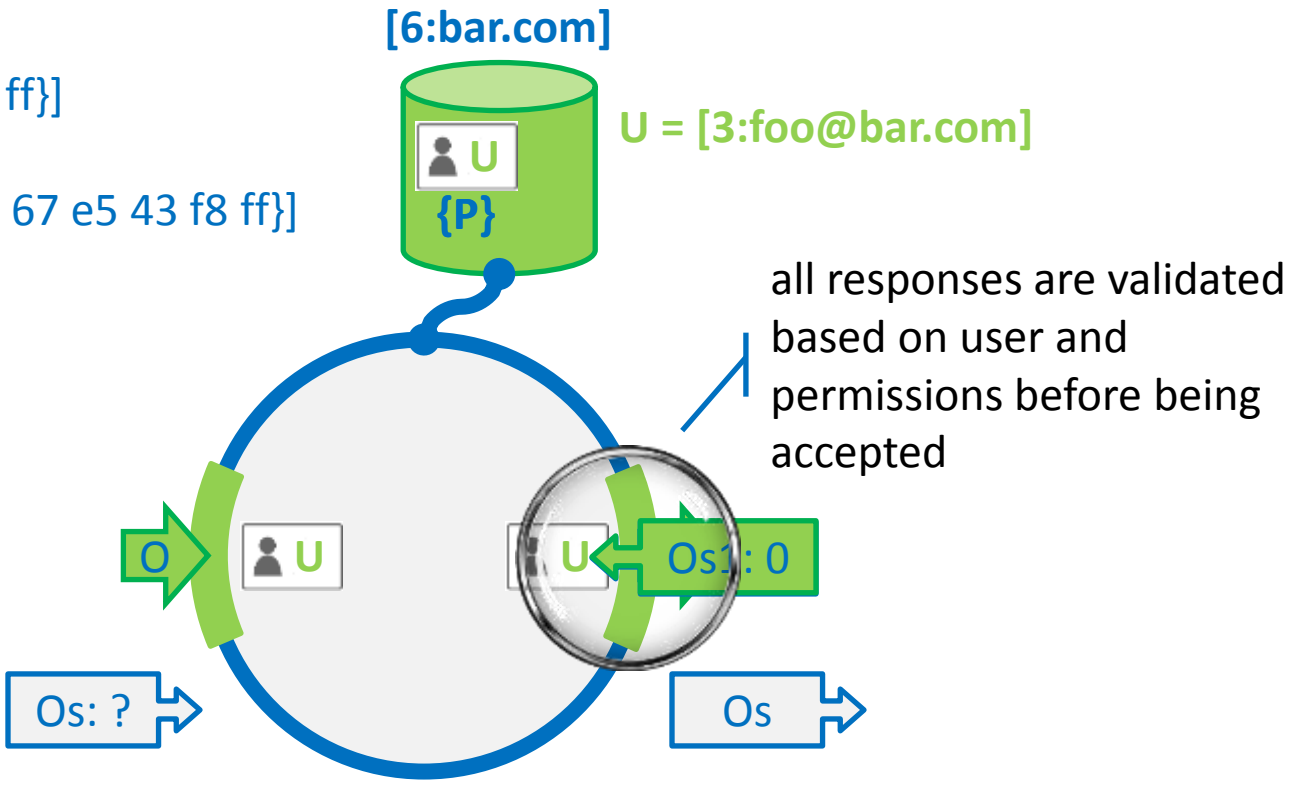
Result

O = [2:{d0 67 e5 43 f8 ff}]
s = [1:{01}]
Os1 = 1 [1:{01}] [2:{d0 67 e5 43 f8 ff}]



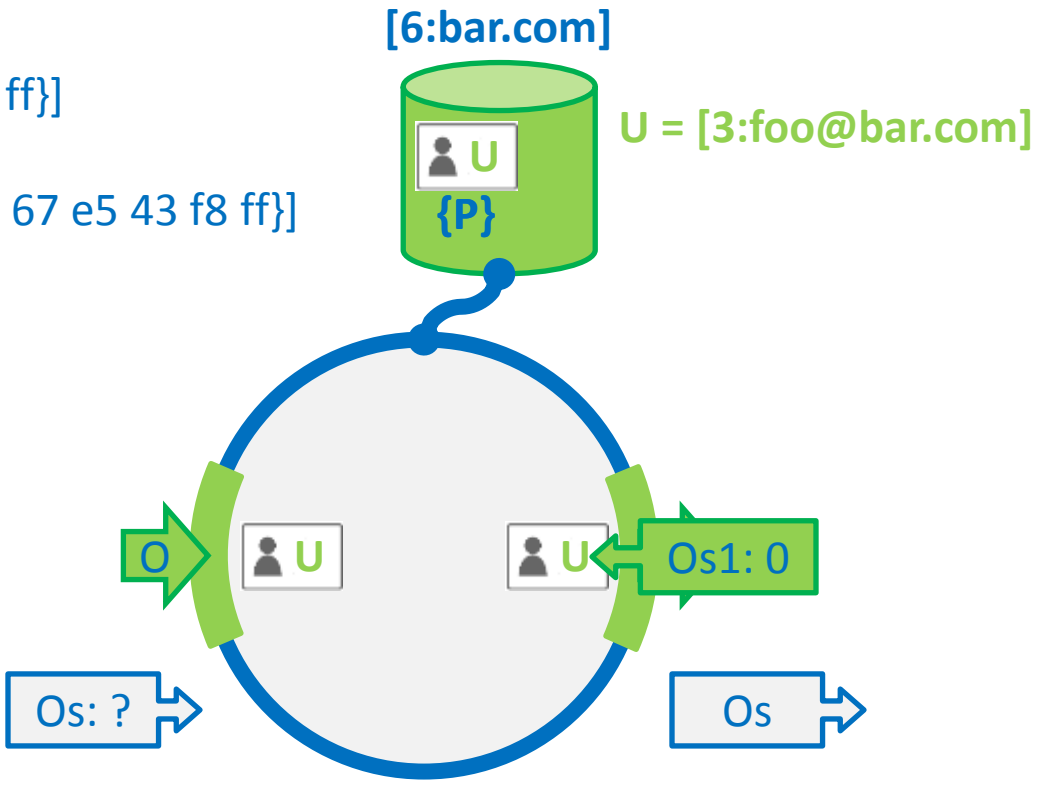
Result

O = [2:{d0 67 e5 43 f8 ff}]
s = [1:{01}]
Os1 = 1 [1:{01}] [2:{d0 67 e5 43 f8 ff}]



Result

O = [2:{d0 67 e5 43 f8 ff}]
s = [1:{01}]
Os1 = 1 [1:{01}] [2:{d0 67 e5 43 f8 ff}]



Supported Interactions

- Properties
  - Get/Set/Subscribe
- Methods
  - Invoke
- Events
  - Register

- Synchronous and asynchronous

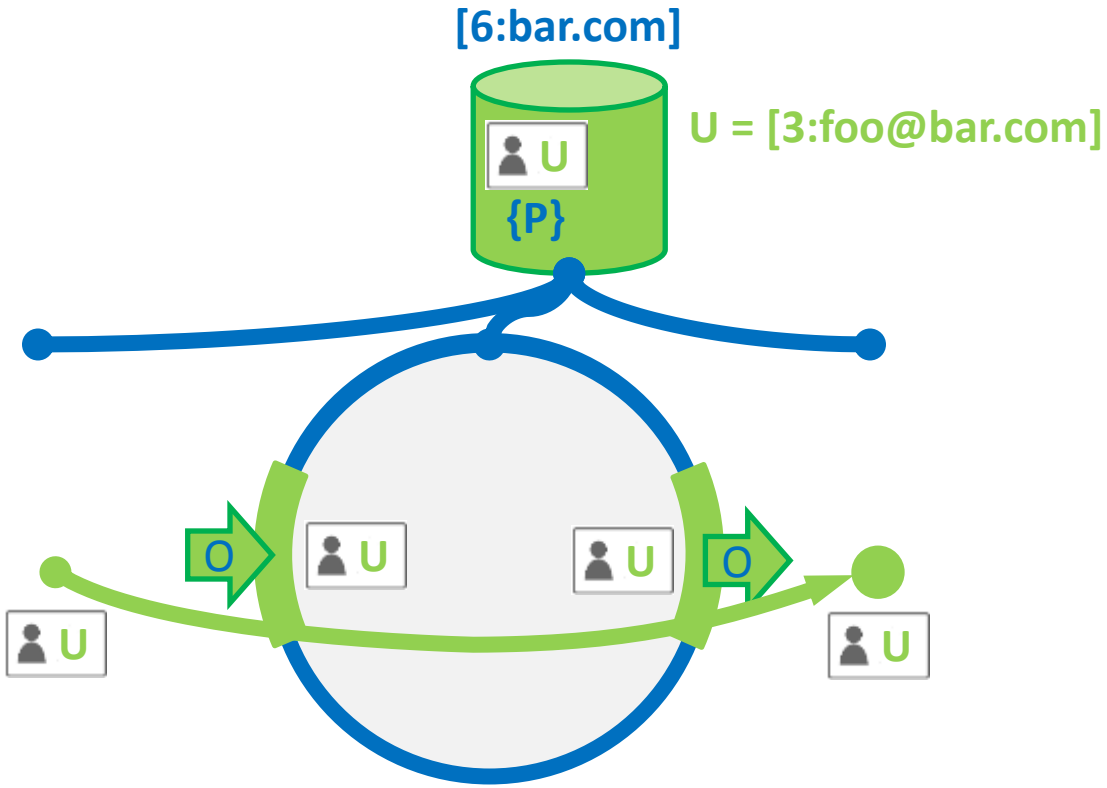
Example – Start A Server

```
#include <dof/oal.h>
DOFServer server;
DOFServerConfigBuilder configBuild;
DOFServerConfig config;
DOFAddress me;
int timeout = 10000;

me = InetTransport_CreateIPv4Address( "0.0.0.0", 3567 );
configBuild = DOFServerConfigBuilder_Create(
                DOFSERVERTYPE_STREAM, me );
DOFServerConfigBuilder_AddCredentials( configBuild, user );
config = DOFServerConfigBuilder_BuildAndDestroy( configBuild );
server = DOF_CreateServer( dof, config );
DOFServer_Start( server, timeout, NULL );
```

Example – Open A Connection

```
import org.opendof.core.oal.*;
DOFConnection connection;
DOFConnection.Config config;
DOFAddress other;
int timeout = 10000;

other = InetTransport.createAddress( "host", 3567 );
config = new DOFConnection.Config.BuildSecureStream( other, user );
connection = dof.createConnection( config );
connection.connect( timeout );
```

Result



What Is Next?

- Scalability to millions of connections
  - Distributed routing problem for discovery
- Optimizations
  - Handling failover for redundant connections
  - Minimizing state updates without too much memory

Questions & Answers