Panasonic IoT Technologies
Panasonic Avionics — A 747 is a ‘Thing’
Panasonic Solutions — A 20,000 Sq. Foot ‘Thing’
Panasonic Automotive – A Car is a ‘Thing’

No. 1 Global Provider of Infotainment Systems

Panasonic

A Better Life, A Better World
Preview Of Coming Attractions...

• DOF
  – Distributed
  – Object
  – Framework
• OpenDOF
  – Open source specifications and implementation
• **Object** is a distributed set of uniquely identified capabilities, bound to an Object Identifier

• **Interface** is a defined set of items (properties, methods, events, exceptions) bound to an Interface Identifier

• **Identity** is a unique persona associated with a secret and permissions

• **Domain** is a centrally managed set of identities
Object Identifiers (OID)

- Globally unique, no registration required
  - Class ID (compressed 32 bit, $2^{30}$ space) is registered
- Unique data (depends on class ID, 63 byte max. length)
- Optional attributes (attribute ID + length + data, 257 byte max. length)
- Maximum OID length = $68 + 128 \times 257$
Object Identifiers (OID)

[3: bryant.eastham@us.panasonic.com]
03 1F 62 72 79 61 6e 74 2e 65 61 73 74 68 61 6d 40 75 73 6f 6e 69 63 2e 63 6f 6d

[2: {d0 67 e5 43 f8 ff}]
02 06 d0 67 e5 43 f8 ff
Interface Identifiers (IID)

- Globally unique through registration
- Registry ID (6 bits, 2 values reserved, 2 assigned, 60 available)
- Length flag (1 byte, 2 byte, 4 byte)
- Data
- 5 byte max. length
Interface Identifiers (IID)

[1:{01}]
05 01

[2:{01 07}]
0A 01 07
Item Identifiers (ItemID)

• Unique within a single interface
• Identifier (compressed 16 bit, $2^{15}$ possible)
• Represents an item type and data type
  – Property, Method, Event, Exception
  – Includes semantic and syntactic information
Putting It All Together

- **Bindings** are OID plus IID
- Operations usually require binding plus 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)
Unified Key Distribution And Access Control

- Nonce is ‘unique’, but arbitrary length
- Ticket is flexible
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
import org.opendof.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 );
Result

\[ U = [3:\text{foo@bar.com}] \]

- **domain data** (centralized)
- **domain identifier**
- **identity**

System 1

System 2

User

---

A Better Life, A Better World
API Example – Instantiate An Object

import org.opendof.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

\[ U = [3:foo@bar.com] \]

A Better Life, A Better World
import org.opendof.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 ) );
        }
    }
}
import org.opendof.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, new QueryListener() );

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

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

\[ s = [1:\{01\}] \]

\[ U = [3:\text{foo@bar.com}] \]

\[ U = [3:\text{foo@bar.com}] \]

\[ [6:\text{bar.com}] \]

<table>
<thead>
<tr>
<th>Query Listener</th>
<th>Provide Listener</th>
</tr>
</thead>
<tbody>
<tr>
<td>Os: ?</td>
<td>Os</td>
</tr>
</tbody>
</table>

Panasonic

A Better Life, A Better World
import org.opendof.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

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

\[ U = [3:foo@bar.com] \]
all interactions are validated based on user and permissions before being accepted

users, credentials, and permissions are centrally stored and managed
Result

\[ O = [2:\{d0 \ 67 \ e5 \ 43 \ f8 \ ff\}] \]
\[ s = [1:\{01\}] \]
\[ Os_1 = 1 \ [1:\{01\}] \ [2:\{d0 \ 67 \ e5 \ 43 \ f8 \ ff\}] \]

\[ U = [3:\text{foo@bar.com}] \]

\[ Os_1: ? \rightarrow U \]
\[ Os: ? \rightarrow U \]
\[ Os \rightarrow Os_1: 0 \]

\[ [6:\text{bar.com}] \]
Result

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

s = [1:{01}]

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

U = [3:foo@bar.com]

all responses are validated based on user and permissions before being accepted
Result

\[ O = [2:\{d0, 67, e5, 43, f8, ff\}] \]
\[ s = [1:\{01\}] \]
\[ Os1 = 1 [1:\{01\}] [2:\{d0, 67, e5, 43, f8, ff\}] \]
import org.opendof.*;
DOFServer server;
DOFServer.Config config;
DOFAddress me;
int timeout = 10000;

me = InetTransport.createAddress( "0.0.0", 3567 );
config = new DOFServer.Config.BuildSecureStream( me, user );
server = dof.createServer( config );
server.start( timeout );
import org.opendof.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

\[ U = [3:foo@bar.com] \]

A Better Life, A Better World
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