Service Layer API for oneM2M

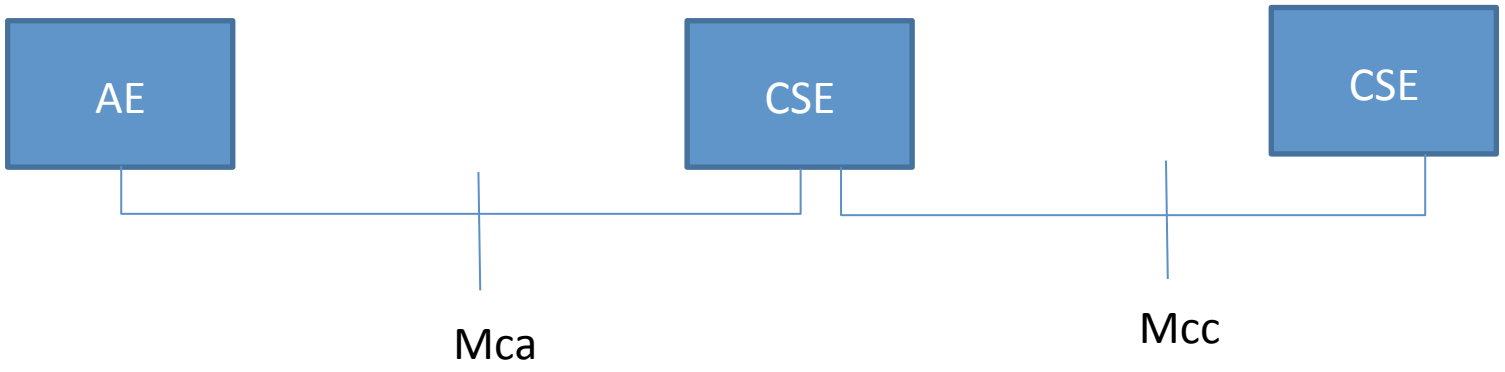
2017/4/18

Hiroyuki Maeomichi

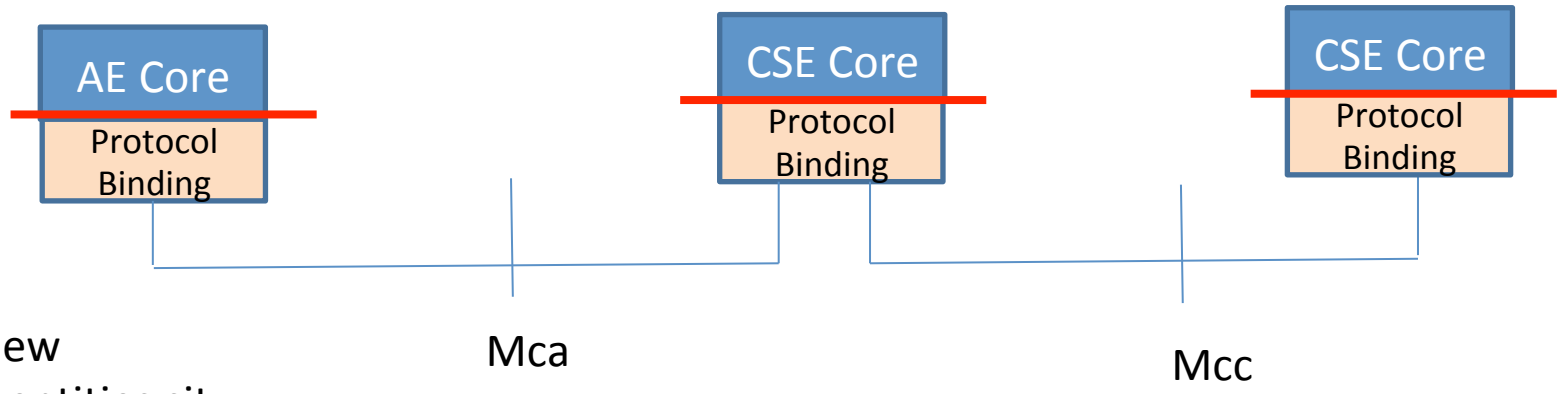
Topics

- DTO design
- Validator
- Sequence form Application
- Cardinality, Permission
- High level API

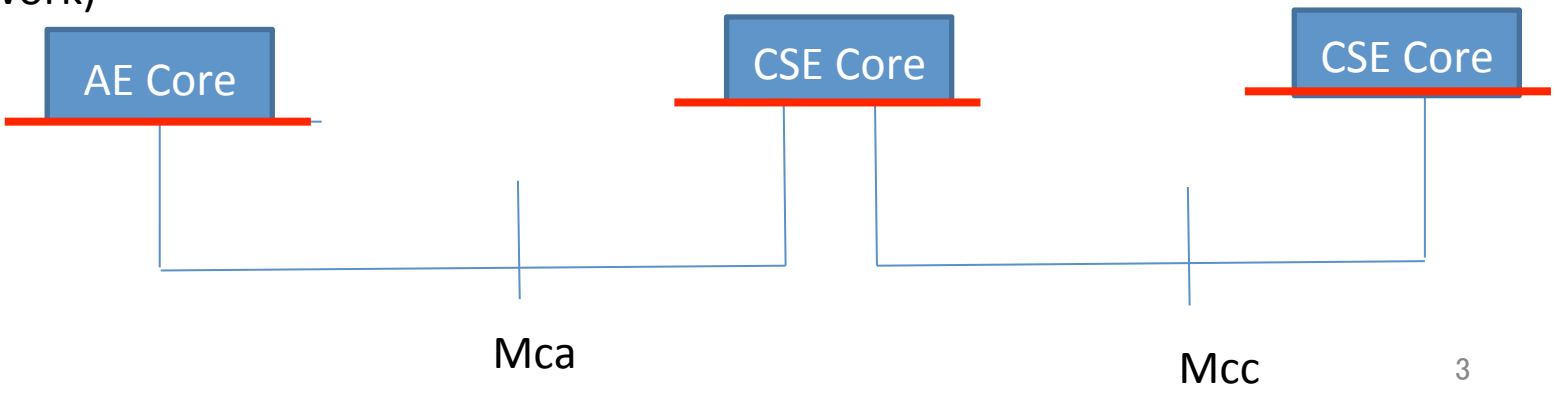
oneM2M view



OSGi RFC237 View



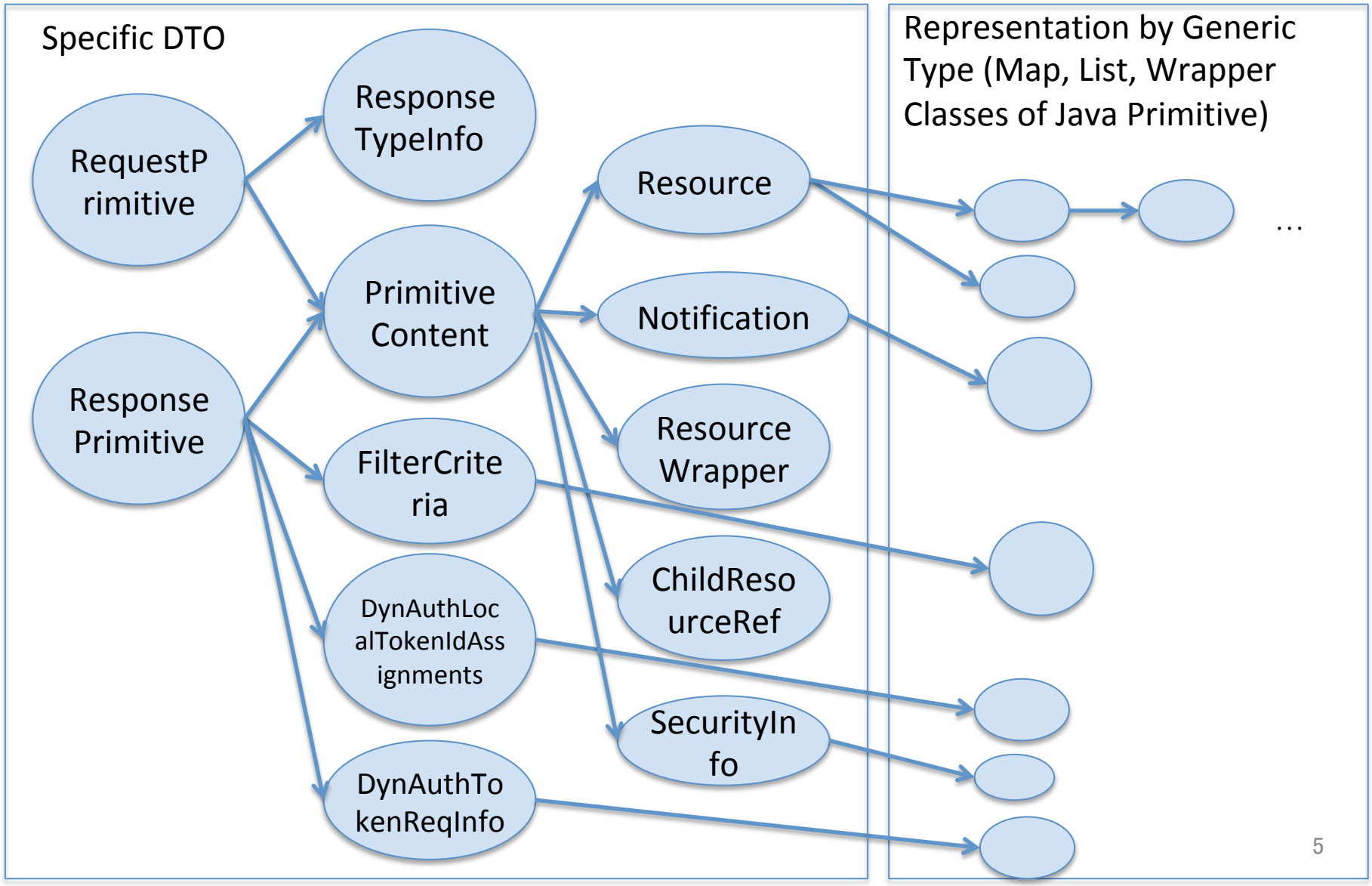
OSGi RFC237 View
(In case that all entities sit in same framework)



Service Layer IF

```
public interface ServiceLayer {
  /**
  * send a request.
  *
  * @param request request
  * @return promise for ResponseDTO.
  */
Promise<ResponseDTO> request(RequestDTO request);
}
```

Specific DTO Types and Generic Type



RequestPrimitiveDTO

```
package org.osgi.service.onem2m.newdto;
public class RequestPrimitiveDTO extends
org.osgi.dto.DTO{
        public Integer operation;
        public String to;
        public String from;
        public String requestIdentifier;
        public Integer resourceType;
        public PrimitiveContentDTO primitiveContent;
        public java.util.List<String> roleIDs;
        public String originatingTimestamp;
        public String requestExpirationTimestamp;
        public String resultExpirationTimestamp;
        public String operationExecutionTime;
        public ResponseTypeInfoDTO responseType;
        public String resultPersistence;
        public Integer resultContent;
        public String eventCategory;
        public Boolean deliveryAggregation;
        public String groupRequestIdentifier;
        public FilterCriteriaDTO filterCriteria;
        public Integer discoveryResultType;
        public String tokens;
        public String tokenIDs;
        public java.util.List<String> localTokenIDs;
        public Boolean tokenReqIndicator;
}
```

ResponsePrimitiveDTO

```
public class ResponsePrimitiveDTO extends org.osgi.dto.DTO{
    public Integer responseStatusCode;
    public String requestIdentifier;
    public PrimitiveContentDTO primitiveContent;
    public String to;
    public String from;
    public String originatingTimestamp;
    public String resultExpirationTimestamp;
    public String eventCategory;
    public Integer contentStatus;
    public Integer contentOffset;
    public DynAuthLocalTokenIdAssignmentsDTO assignedTokenIdentifiers;
    public DynAuthTokenReqInfoDTO tokenReqInfo;
}
```

Design Choice:
Mandatory field can be primitive.
Wrapper was chosen for unity to other part.

PrimitiveContentDTO

```
public class PrimitiveContentDTO extends org.osgi.dto.DTO{
        // only one of following fields is assigned.
        ResourceDTO resource;
        NotificationDTO NotificationDTO;
        List<NotificationDTO> aggregatedNotification;
        SecurityInfoDTO securityInfo;
        ResponsePrimitiveDTO responsePrimitive;

        ResourceWrapperDTO resourceWrapper;
        String debugInfo;
        List<String> listOfURIs;
        String uri;
        List<ResponsePrimitiveDTO> aggregatedResponse;
        List <ChildResourceRefDTO> childResourceRefList;
}
```

Design Choice:
Include parameter indicates which field are assigned. Chose simpler one, because it makes no big difference.

Design Choice:
Uri and NotificationDTO seems to be merged with List of it. Choose this form in case for distinguishing those patterns.

NotificationDTO

```
public class NotificationDTO extends org.osgi.dto.DTO{
    public NotificationEventDTO notificationEvent;
    public Boolean verificationRequest;
    public Boolean subscriptionDeletion;
    public String subscriptionReference;
    public String creator;
    public String notificationForwardingURI;
    public Map<String, Object> ipeDiscoveryRequest;
}
```

ResourceDTO

```
public final class ResourceDTO extends org.osgi.dto.DTO{
        // Universal Attribute, which can be held by all resources.
        public Integer resourceType;
        public String resourceID;
        public String parentID;
        public String creationTime;
        public String lastModifiedTime;
        public String resourceName;

        // optional, Universal Attributes
        public java.util.List<String> labels;

        //Non Universal Attribute.
        //Though value part is show as Object, but it must be in types that are allowed for OSGi DTO.
        public Map<String, Object> attribute;
}
```

Design Choice:
Mandatory field can be primitive.
Wrapper was chosen for unity to other part.

Design Choice:
All parameters can be put in *attribute* map.
This form is more specific.

FilterCriteriaDTO

```
public class FilterCriteriaDTO extends org.osgi.dto.DTO{
    public String createdBefore;
    public String createdAfter;
    public String modifiedSince;
    public String unmodifiedSince;
    public Integer stateTagSmaller;
    public Integer stateTagBigger;
    public String expireBefore;
    public String expireAfter;
    public java.util.List<String> labels;
    public java.util.List<String> resourceType;
    public Integer sizeAbove;
    public Integer sizeBelow;
    public java.util.List<String> contentType;
    public java.util.List<Map> attribute;
    public Integer filterUsage;
    public Integer limit;
    public java.util.List<String> semanticsFilter;
    public Boolean filterOperation;
    public Integer contentFilterSyntax;
    public String contentFilterQuery;
    public Integer level;
    public Integer offset;
}
```

Note:
Optional fields. Wrapper is possible to express 'Not present' by null.

ResponseTypeInfoDTO

```
public class ResponseTypeInfoDTO extends
org.osgi.dto.DTO{
    public Integer responseTypeValue;
    public List<String> notificationURI;
}
```

ResourceWrapperDTO

```
public class ResourceWrapperDTO extends
org.osgi.dto.DTO{
    public String uri;
    ResourceDTO resource;
}
```

DynAuthLocalTokenIdAssignmentsDTO

```
public class DynAuthLocalTokenIdAssignmentsDTO extends
org.osgi.dto.DTO{
    java.util.List<Map> localTokenIdAssignment;
}
```

DynAuthTokenReqInfoDTO

```
public class DynAuthTokenReqInfoDTO extends
org.osgi.dto.DTO{
        public java.util.List<Map> dasInfo;
}
```

ChildResourceRefDTO

```
public class ChildResourceRefDTO extends
org.osgi.dto.DTO{
    public String uri;
    public String name;
    public Integer type;
    public String specializationID;
}
```

SecurityInfoDTO

```
public class SecurityInfoDTO extends org.osgi.dto.DTO{
        public Integer securityInfoType;
        public Map dasRequest;
        public Map dasResponse;
        public Map esprimRandObject;
        public String esprimObject;
        public byte[] escertkeMessage;
}
```

Validator

Validator

- In previous teleco, IF was like..

  boolean isValid(RequestDTO req);
- Changed to tell problems to caller.

```
public interface Introspector {
    public String[] findValidationProblems(ResponsePrimitiveDTO resp);
    public String[] findValidationProblems(RequestPrimitiveDTO req);
    public String[] findValidationProblems(ResourceDTO resource);

...
```

Retrieval of Attribute Names

```
public interface Introspector {
    ...
    public String[] getAttributeNames(int resourceType);

    public String[] getMandatoryAttributesForCreate(int resourceType);
    public String[] getOptionalAttributesForCreate(int resourceType);
    public String[] getNotPresentAttributesForCreate(int resourceType);

    public String[] getMandatoryAttributesForUpdate(int resourceType);
    public String[] getOptionalAttributesForUpdate(int resourceType)
    public String[] getNotPresentAttributesForUpdate(int resourceType);
    ...
}
```

Retrieval of Types

getTemplateObject() is introduced because getType() is useless for List or Map.

```
public interface Introspector {
    …
    public Class getType(int resourceType, String Attribute);
    public Object getTemplateObject(int resouceType, String Attribute);
    …
}
```

Example of Template Object
List → Map. keyA → List-> String
keyB → Map.keyC → Integer
keyD -> String

Validation for User Defined Type

Introduction of FlexContainer of oneM2M:

- FlexContainer Resource can be defined by user with adding custom attributes. Initially it was intended to be like JSON structure in oneM2M world.
- The definition of custom flex container is expressed in ‘contentDefinition’ attribute of the FlexContainer. It include URI of XSD definition. The XSD content is stored as contentInstance Resource in some CSE.

Options of FlexContainer validator

1. No support.
2. Offline support(Support by Hand)
   Developers make extra validator function after reading XSD files. After that, a Validator Service is instantiated on FW with properties of ‘supportedContainerDefinition’.
3. Online Support (Explicit):
   AE fetches XSD and pass to Validator. Validator analyses the XSD and support.
4. Online Support (Implicit):

   After method call on Validator, Validator will get the XSD as Applications. After analysis of the XSD, Validator checks data format based on that.

Designed I/F based on option 2.

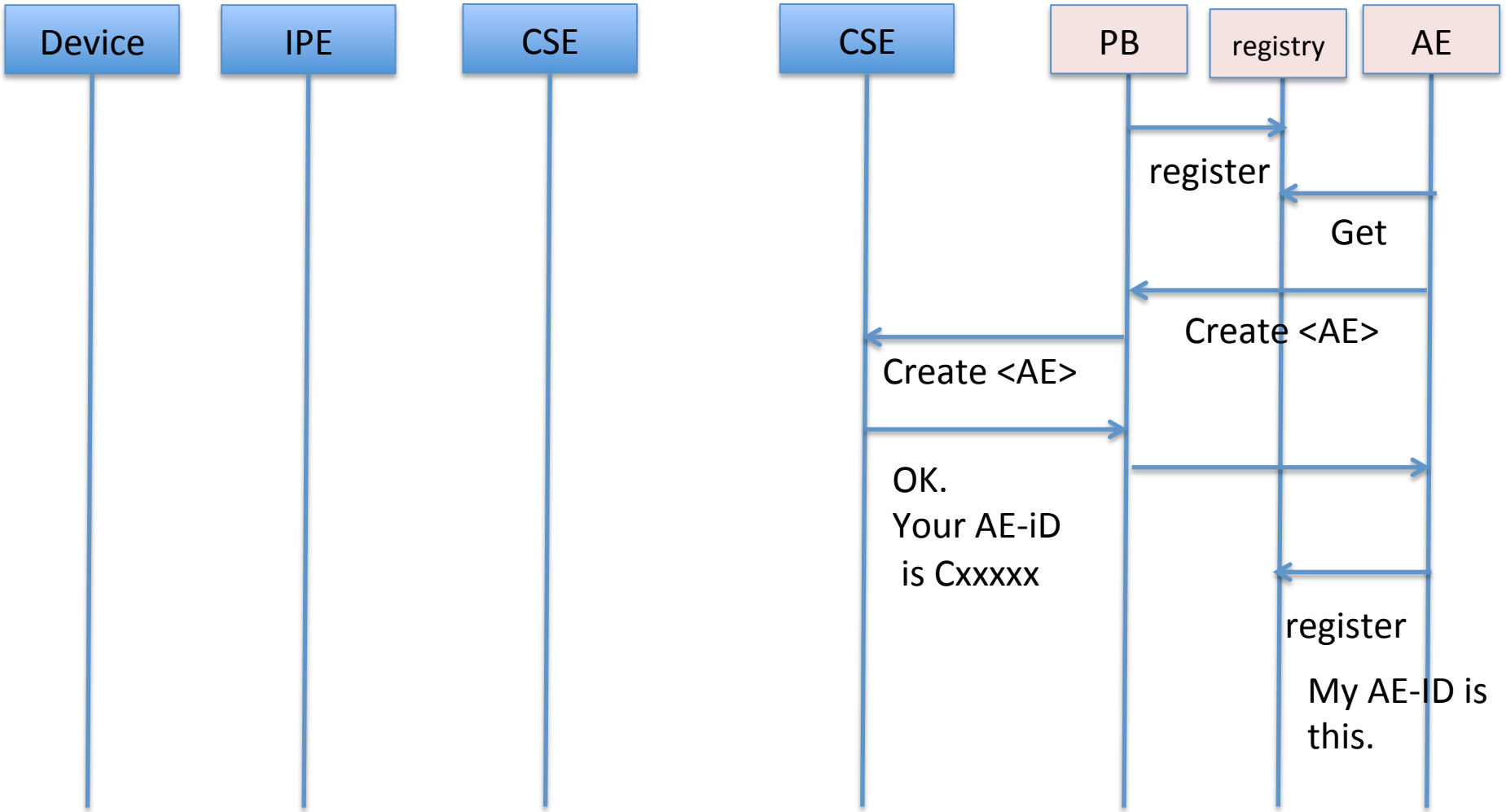
FlexContainerIntrospector IF

```
public interface FlexContainerIntrospector {
    public String[] findValidationProblems(ResourceDTO
resource);
    public String[] getCustomeAttributeNames(String
containerDefinition);
    public Class getType(String containerDefinition, String
customAttributeName);
    public Object getTemplateObject(String
containerDefinition, String customAttributeName);
}
```

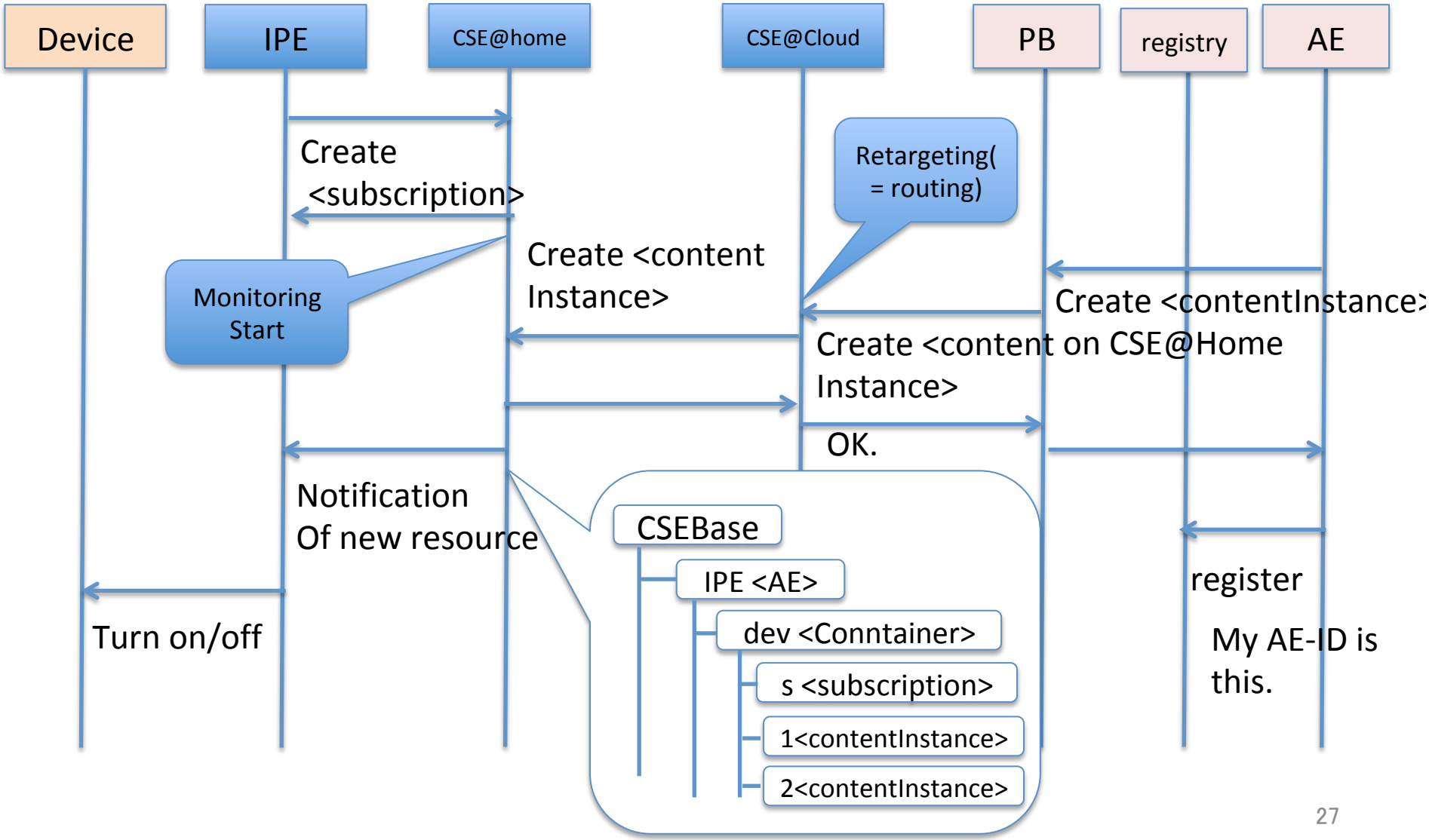
This service expose supporting definition as property.

Sequence

Sequence of Registration



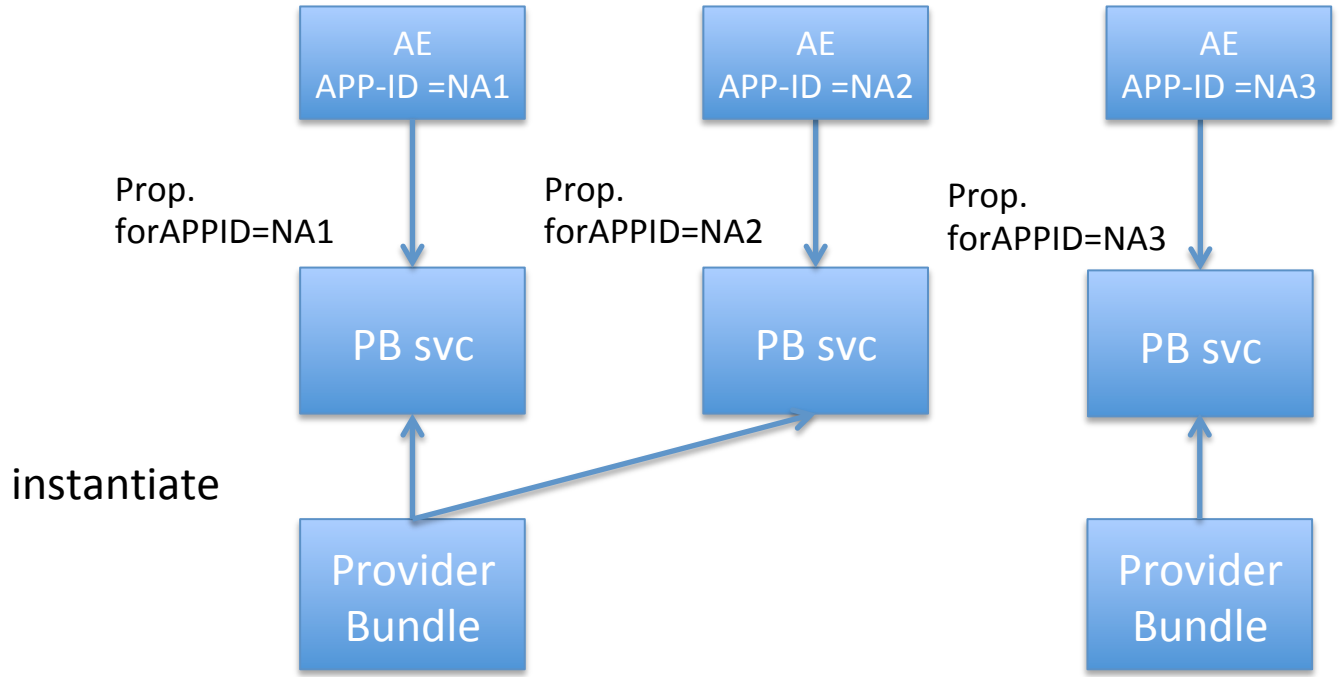
Sequence of Device Control



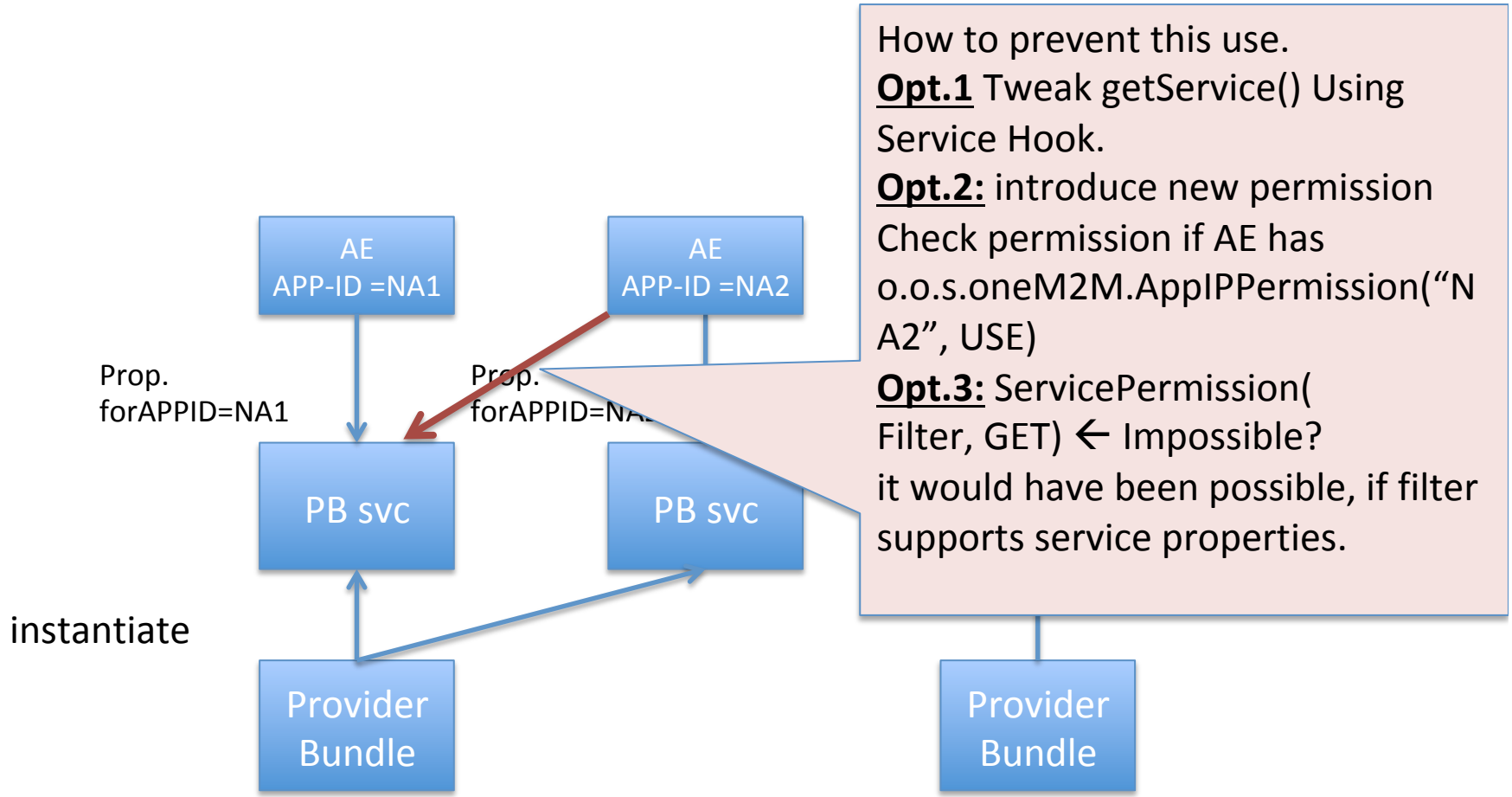
Bootstrap

- Proposing Model
  - Multiple bundles create ‘ProtocolBinding’ Services, with different configurations.
    - Hosting CSE (=connecting CSE)
    - Using Protocol: HTTP, MQTT, CoAP, WS
    - Using Serialization: XML, JSON, CBOR
    - Use of secure connection: like HTTPS
    - Config for secure connection: Certificate, key…
    - Put ‘APP-ID’ property for specifying intended user (AE).
  - AE pick up ProtocolBinding Service by using ‘APP-ID’, which is encoded in code.
  - It is administrator’s responsibility to prepare necessary ProtocolBinding services by configuring provider bundles. How to configure is out of scope of specification, because it would be very diverse.

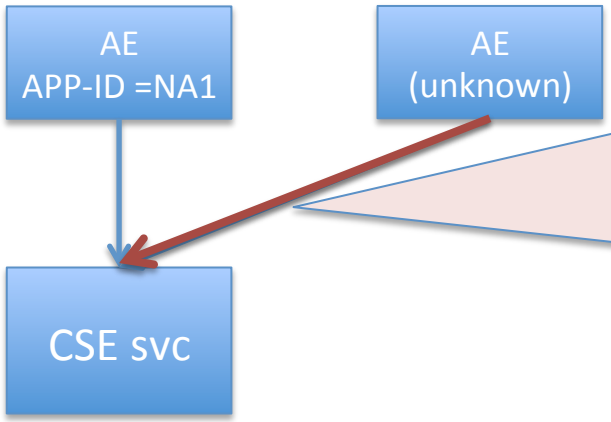
Bootstrap: Cardinality



Control of unintended use (AE -> PB)



Control of unintended use (AE -> CSE)



How to prevent this use.

Opt.1 Tweak getService() Using Service Hook.

Opt.2: conrol by ServicePermission (“org.osgi.service.onem2m.Cse “, “GET”)

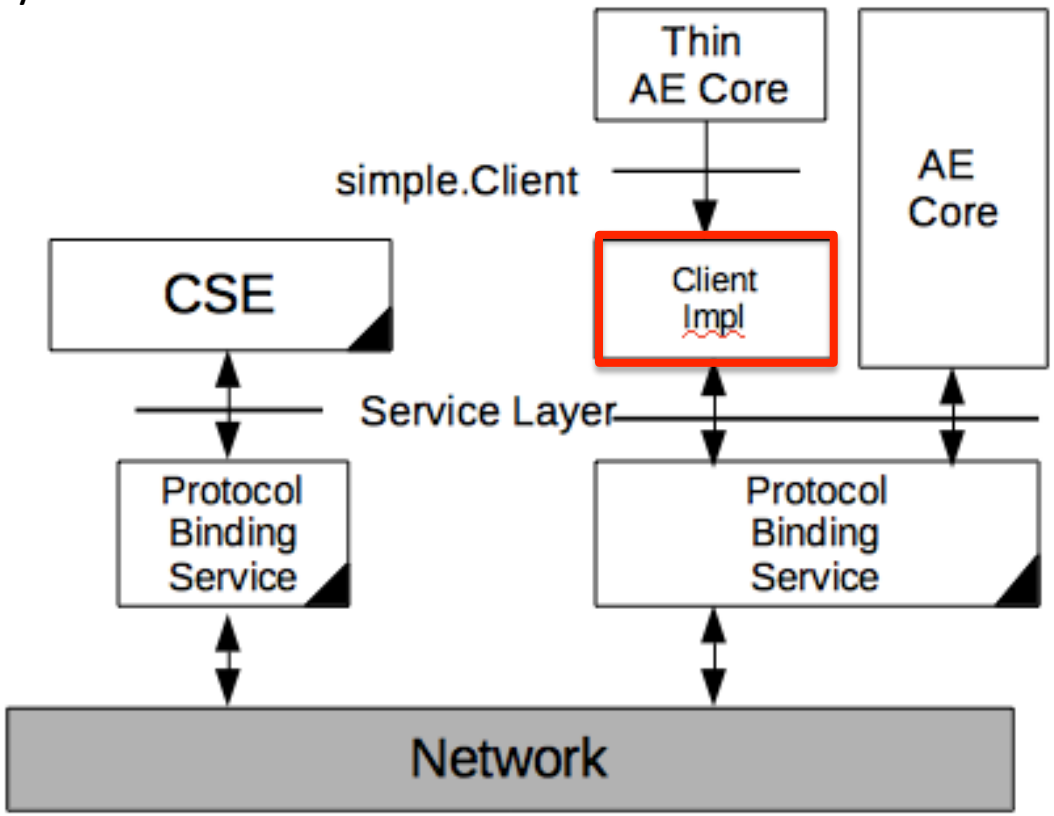
Client Library

Client library

Client Interface provides higher level api to application for easier development. It

1) provids CRUD operations granularity and hides detail data structure of RequestPrimitive and ResponsePrimitive, and
2) automatically assigns some protocol parameters like *request ids* or *from* field.

It is assumed very small code around 100～200 lines.

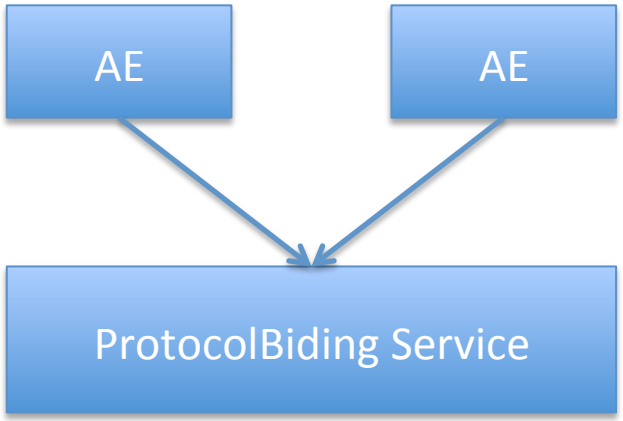


Overview of API

```
package org.osgi.service.onem2m.simple;
public interface Client {
        public ResourceDTO create(String uri, ResourceDTO resource) ;
        public ResourceDTO retrieve(String uri) ;
        public ResourceDTO retrieve(String uri, String[] nesessaryAttributeNames);
        public ResourceDTO update(String uri, ResourceDTO resource);
        public void delete(String uri );
        public List<String> discover(String uri, FilterCriteriaDTO fc);
        public void notify(NotificationDTO notification);
}
package org.osgi.service.onem2m.simple.impl;
public class ClientImpl implements Client {
        public ClientImpl(String appid, String aeid, NotificationListener listener) {}
        public ClientImpl(String appid, String aeid, NotificationListener listener, Client customizer) {}
       ..
}
```

Backup slides

Cardinality



For non-secure association,
PB service can be shared by AEs