Tizen/Artik IoT Lecture Chapter 15. IoTivity Notification Service

Sungkyunkwan University

Contents

- **Notification Service**
  - Revisit the Observe Usage in Simple Server
  - Scenario
  - Architecture
- **Operations**
  - Start or Stop Notification service
  - Discovery of Notification Resource
  - Subscription to Notification Resource
  - Sending Notify
  - Synchronization of Notification Message
- **Sample**

Revisit the Usage of OBSERVE in Simple Server

- **Observe Registration and notify**

① ② Entity Handler check observe message & push observer into observer list

```
if(requestFlag & RequestHandlerFlag::ObserverFlag)
{
    ObservationInfo observationInfo = request->getObservationInfo();
    if(ObserveAction::ObserveRegister == observationInfo.action)
    {
        m_interestedObservers.push_back(observationInfo.obsId);
    }
    else if(ObserveAction::ObserveUnregister == observationInfo.action)
    {
        m_interestedObservers.erase(std::remove(
                                    m_interestedObservers.begin(),
                                    m_interestedObservers.end(),
                                    observationInfo.obsId),
                                    m_interestedObservers.end());
    }
```

③ Create Light Value changed Watcher Thread

```
    startedThread)
    pthread_create (&threadId, NULL, ChangeLightRepresentation, (void *)this);
    startedThread = 1;
}
```

```
OCStackResult OC::OCPlatform::notifyListOfObservers ( OCResourceHandle      resourceHandle,
                                                      ObservationIds &      observationIds,
                                                      const std::shared_ptr< OCResourceResponse > responsePtr
                                                    )
```

```
OCStackResult OC::OCPlatform::notifyAllObservers ( OCResourceHandle resourceHandle )
```

```
void * ChangeLightRepresentation (void *param)
{
    LightResource* lightPtr = (LightResource*) param;

    // This function continuously monitors for the changes
    while (1)
    {
        sleep (3);

        if (gObservation)
        {
            // If under observation if there are any changes to the light resource
            // we call notifyObservors
            //
            // For demostration we are changing the power value and notifying.
            lightPtr->m_power += 10;

            cout << "\nPower updated to : " << lightPtr->m_power << endl;
            cout << "Notifying observers with resource handle: " << lightPtr->getHandle() << endl;

            OCStackResult result = OC_STACK_OK;

            if(isListOfObservers) CheckObserverList
            {
                std::shared_ptr<OCResourceResponse> resourceResponse =
                            {std::make_shared<OCResourceResponse>()};

                resourceResponse->setErrorCode(200);
                resourceResponse->setResourceRepresentation(lightPtr->get(), DEFAULT_INTERFACE);

④               result = OCPlatform::notifyListOfObservers(  lightPtr->getHandle(),
                                                             lightPtr->m_interestedObservers,
                                                             resourceResponse);
            }
            else
            {
                result = OCPlatform::notifyAllObservers(lightPtr->getHandle());
            }

            if(OC_STACK_NO_OBSERVERS == result)
            {
                cout << "No More observers, stopping notifications" << endl;
                gObservation = 0;
            }
        }
    }

    return NULL;
}
```

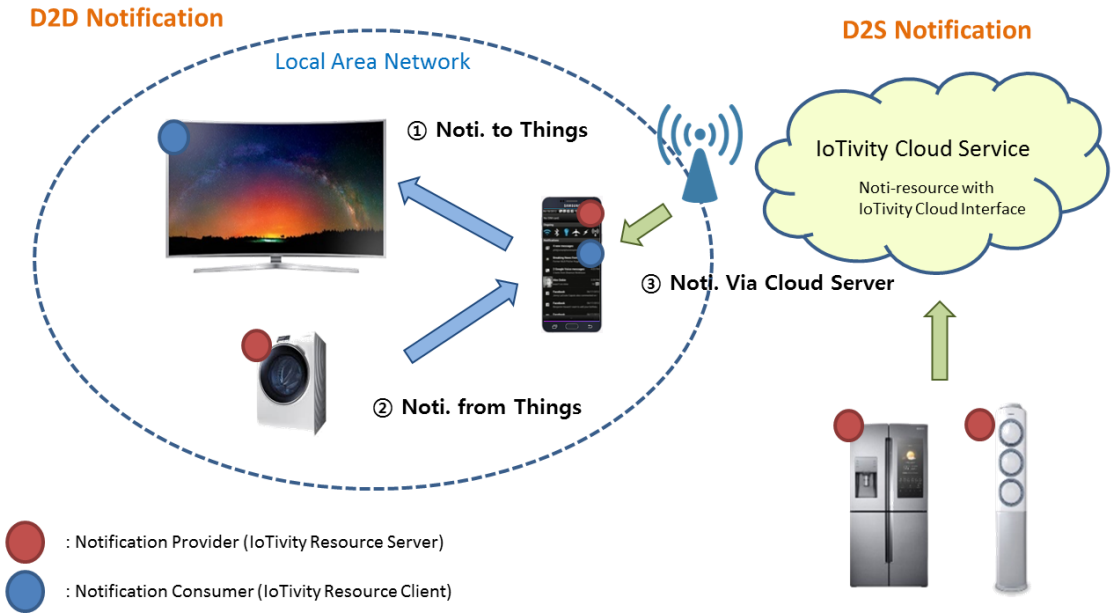
Send Notify

Notification Service

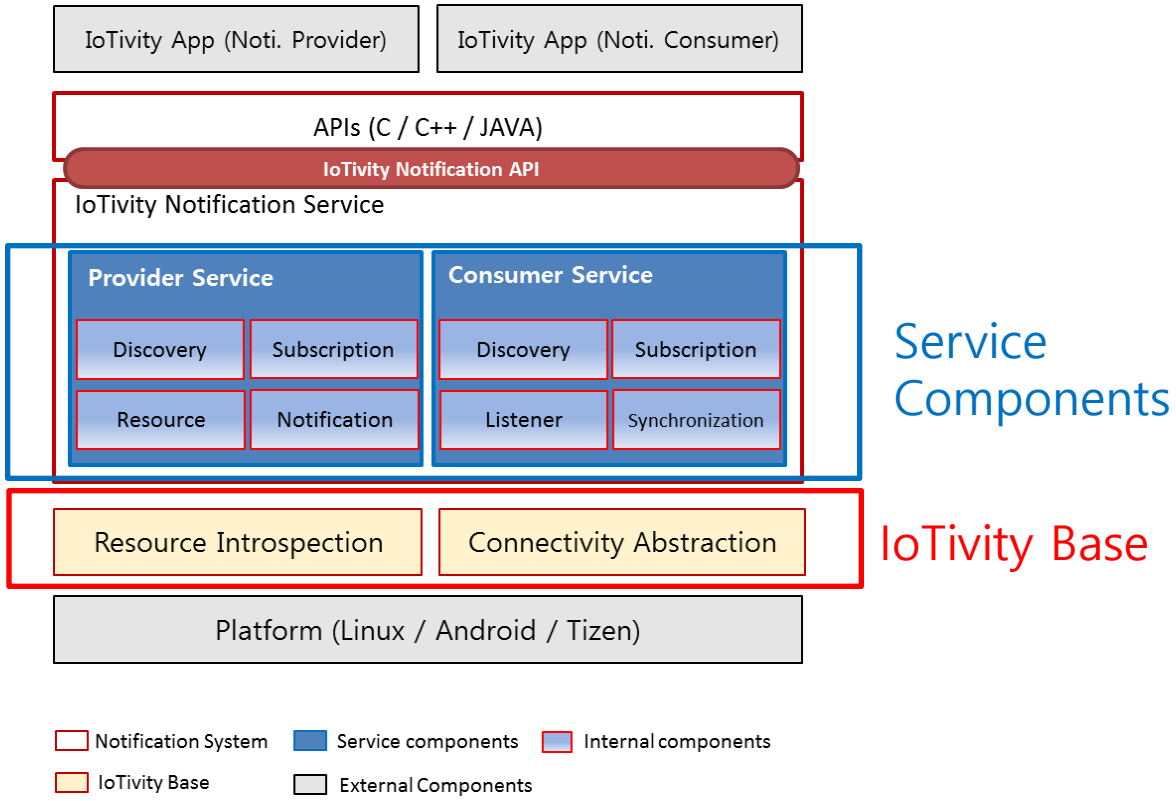
- **It is possible to make notification service only with IoTivity Base API. However…**
  - Notification service can utilize caching and monitoring features
  - Can configure the access policy
  - Topic-based Subscribe and Publish
  - Supports platform independent features
  - Prioritizes the notification
  - Makes fully use of IoTivity Cloud Server
  - etc..

Notification Service Scenario

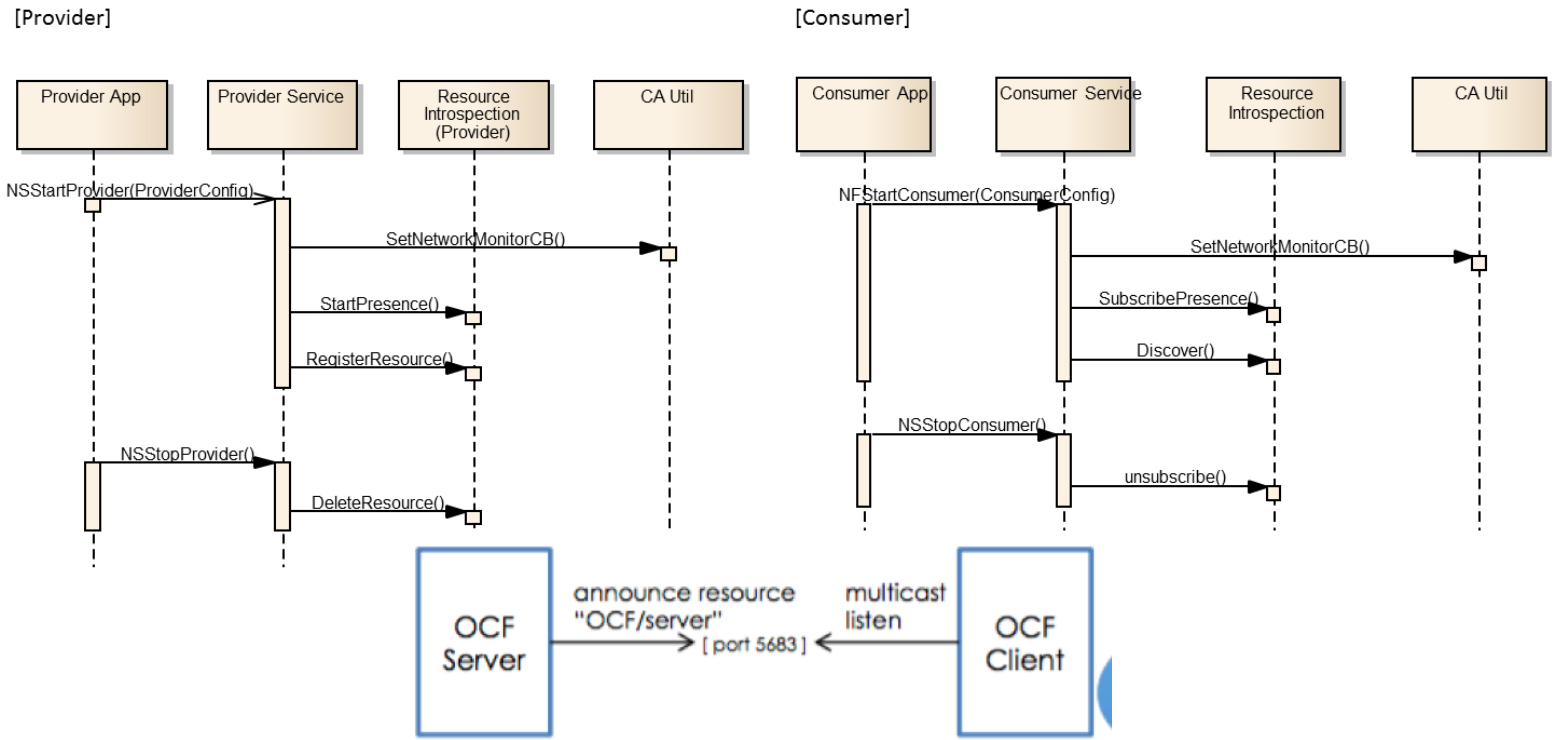
- **D2D Notification - Consumer and Providers are in Local Area Network**
- **D2S Notification - Consumer and Providers are connected via IoTivity cloud**



Notification Service Architecture



Start or Stop Notification service



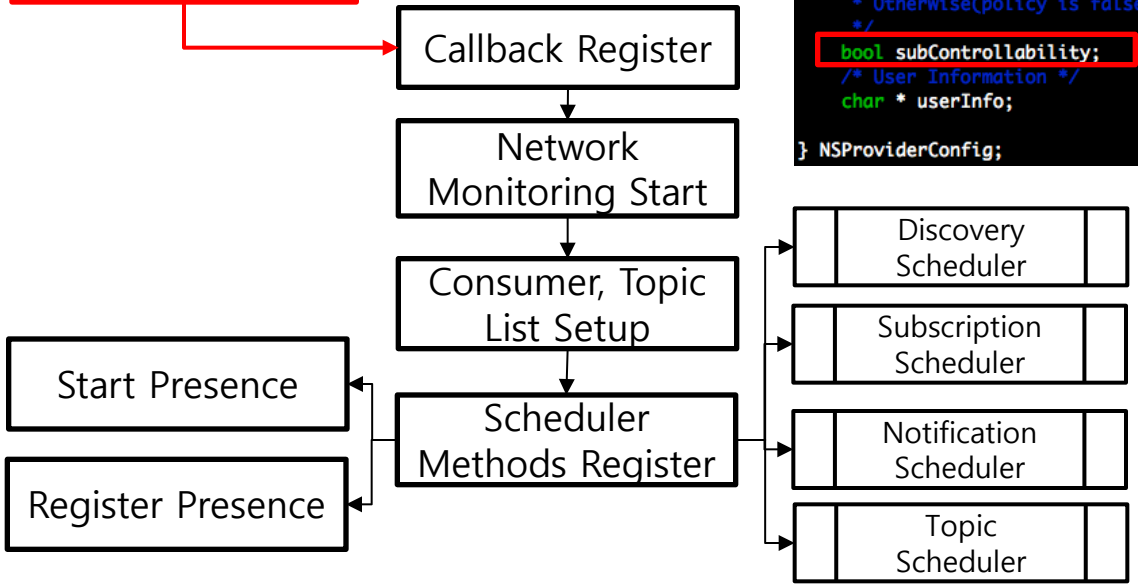
[Figure 1] Multicast announcement over Wi-Fi / Ethernet

Start or Stop - Notification Provider


```
printf("NSStartProvider(Accepter: Provider)");
NSProviderConfig config;
config.subControllability = true;
config.subRequestCallback = subscribeRequestCallback;
config.syncInfoCallback = syncCallback;
config.userInfo = OICStrdup("OCF_NOTIFICATION");
NSStartProvider(config);
```

```
typedef struct
{
    /* Invoked when the subscription request from consumer is received */
    NSSubscribeRequestCallback subRequestCallback;
    /* Invoked when the synchronization data, read and deleted, is sent by consumer is received */
    NSProviderSyncInfoCallback syncInfoCallback;
    /* Set the policy for notification servcie refering to following
     * if policy is true, provider decides to allow or deny for all the subscribing consumers.
     * Otherwise(policy is false) consumer decides to request subscription to discovered providers.
     */
    bool subControllability;
    /* User Information */
    char * userInfo;

} NSProviderConfig;
```

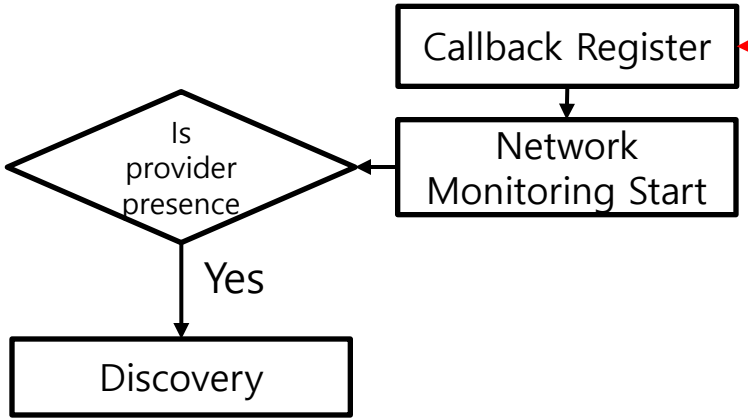
```
if (policy == NS_POLICY_PROVIDER)
{
    NS_LOG(DEBUG, "Place Provider as a subscription accepter");
}
else if (policy == NS_POLICY_CONSUMER)
{
    NS_LOG(DEBUG, "Place Consumer as a subscription accepter");
}
```



Start or Stop - Notification Consumer

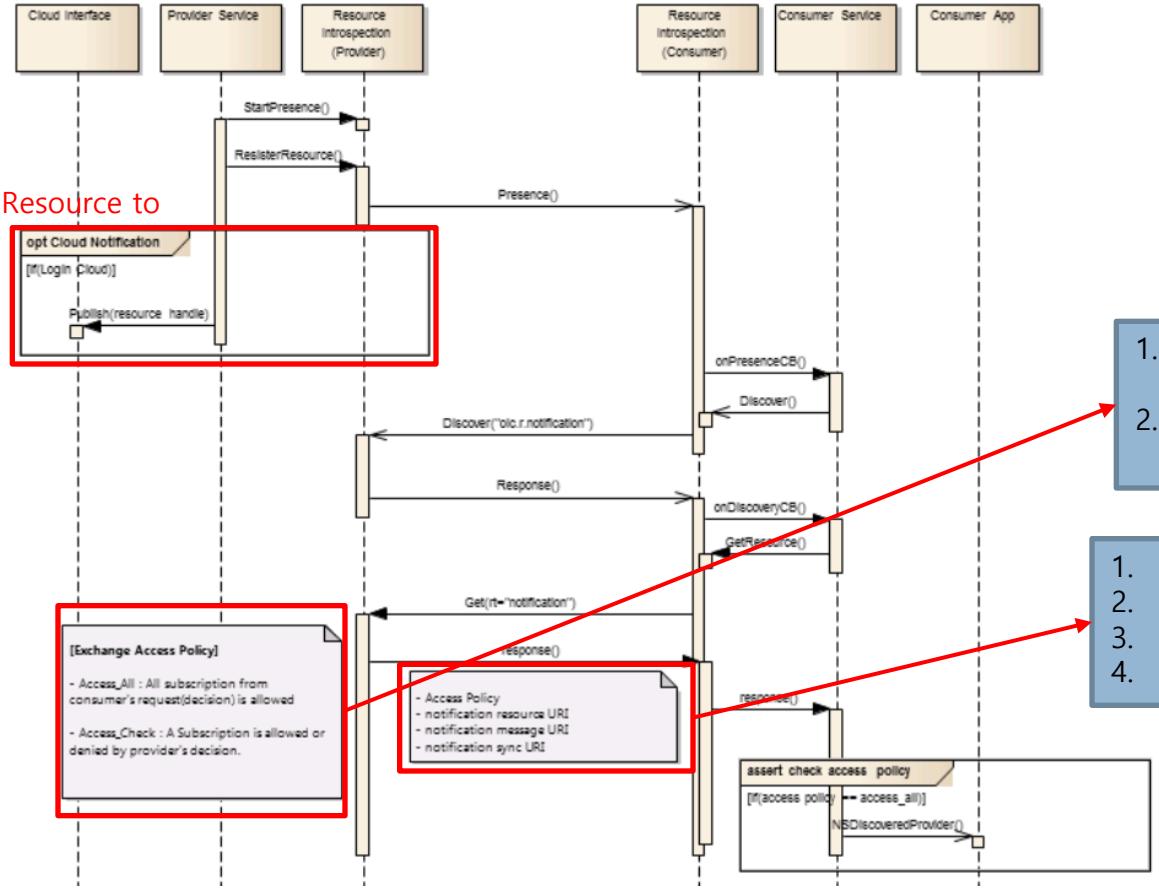

```
        printf("1. Start Consumer\n");
        NSStartConsumer(cfg);
```

```
// Public APIs
NSResult NSStartConsumer(NSConsumerConfig config)
{
    bool isStartedConsumer = NSIsStartedConsumer();
    NS_VERIFY_NOT_NULL(isStartedConsumer == false ? (void *) 1 : NULL, NS_OK);

    NS_VERIFY_NOT_NULL(config.messageCb, NS_ERROR);
    NS_VERIFY_NOT_NULL(config.syncInfoCb, NS_ERROR);
    NS_VERIFY_NOT_NULL(config.changedCb, NS_ERROR);

    NSSetMessagePostedCb(config.messageCb);
    NSSetNotificationSyncCb(config.syncInfoCb);
    NSSetProviderChangedCb(config.changedCb);
    NSSetIsStartedConsumer(true);

    NSResult ret = NSConsumerMessageHandlerInit();
    NS_VERIFY_NOT_NULL_WITH_POST_CLEANING(ret == NS_OK ? (void *) 1 : NULL,
            NS_ERROR, NSStopConsumer());

    return NS_OK;
}
```



Discovery of Notification Resource

Publish Resource to Cloud

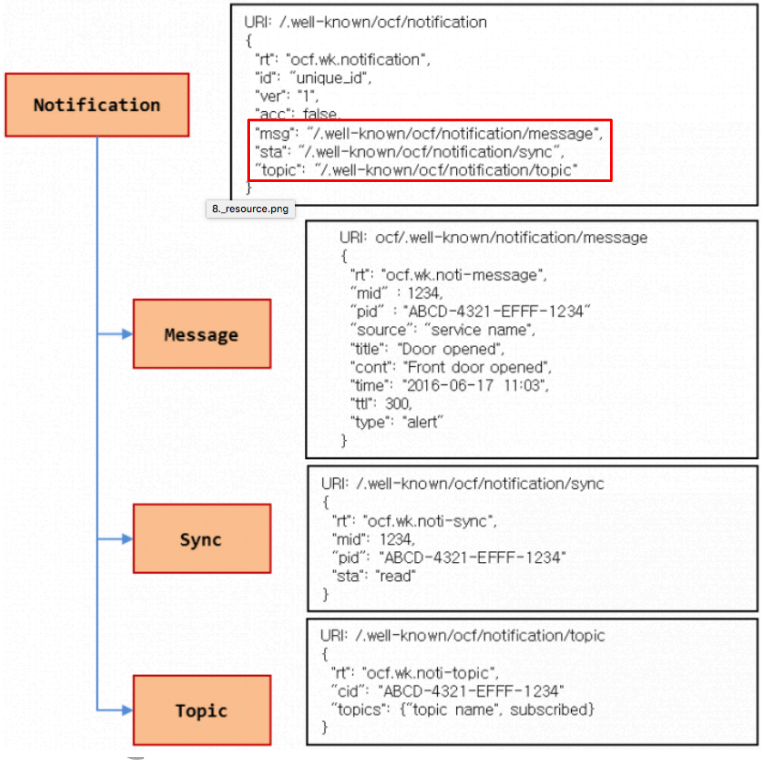


1. Access_all (Accept all subscriptions)
2. Access_Check (Accept or Deny subscriptions)

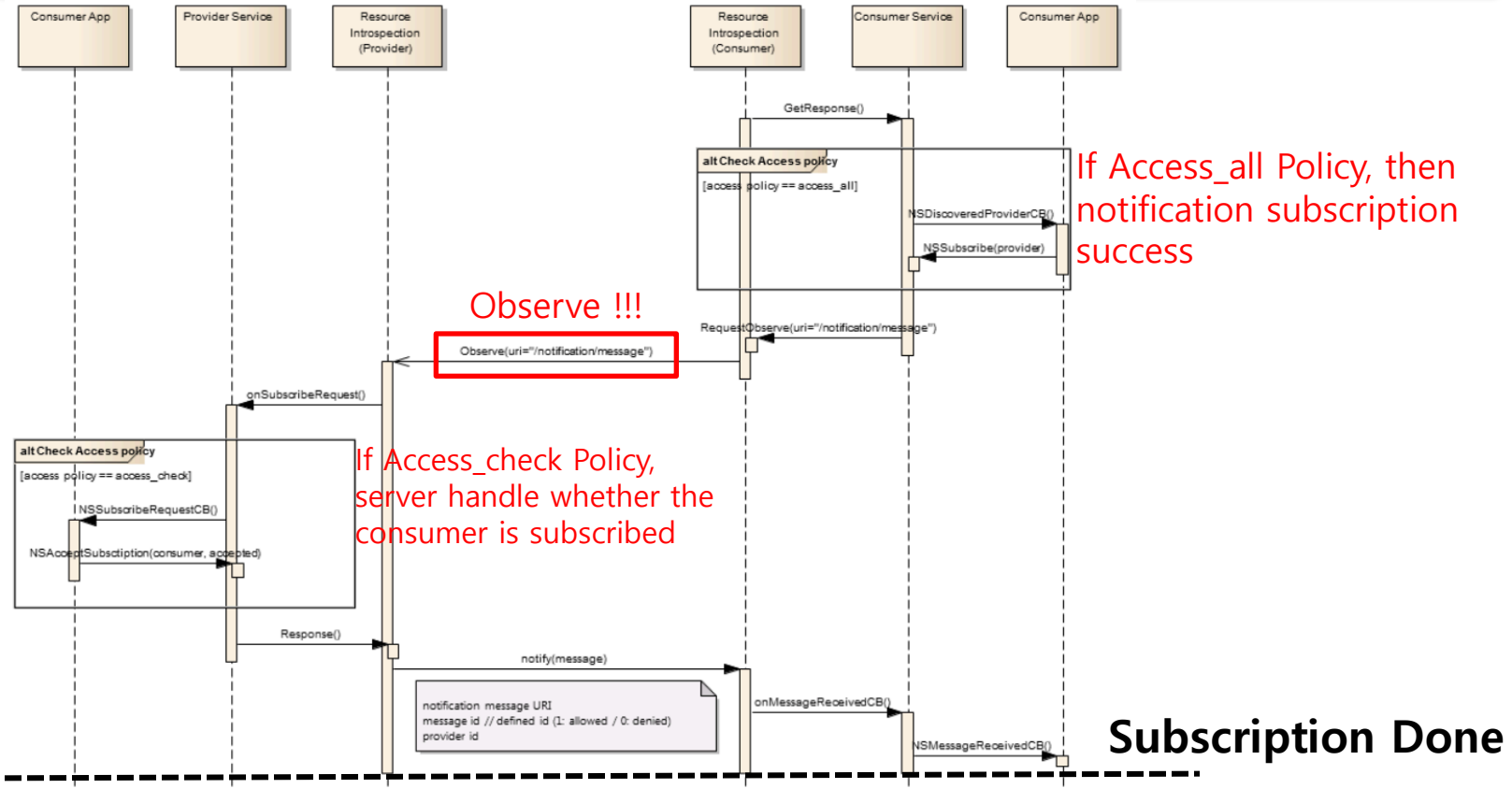
1. Access Policy
2. Notification Resource URI
3. Notification Message(Topic) URI
4. Notification Sync URI

Resource Model

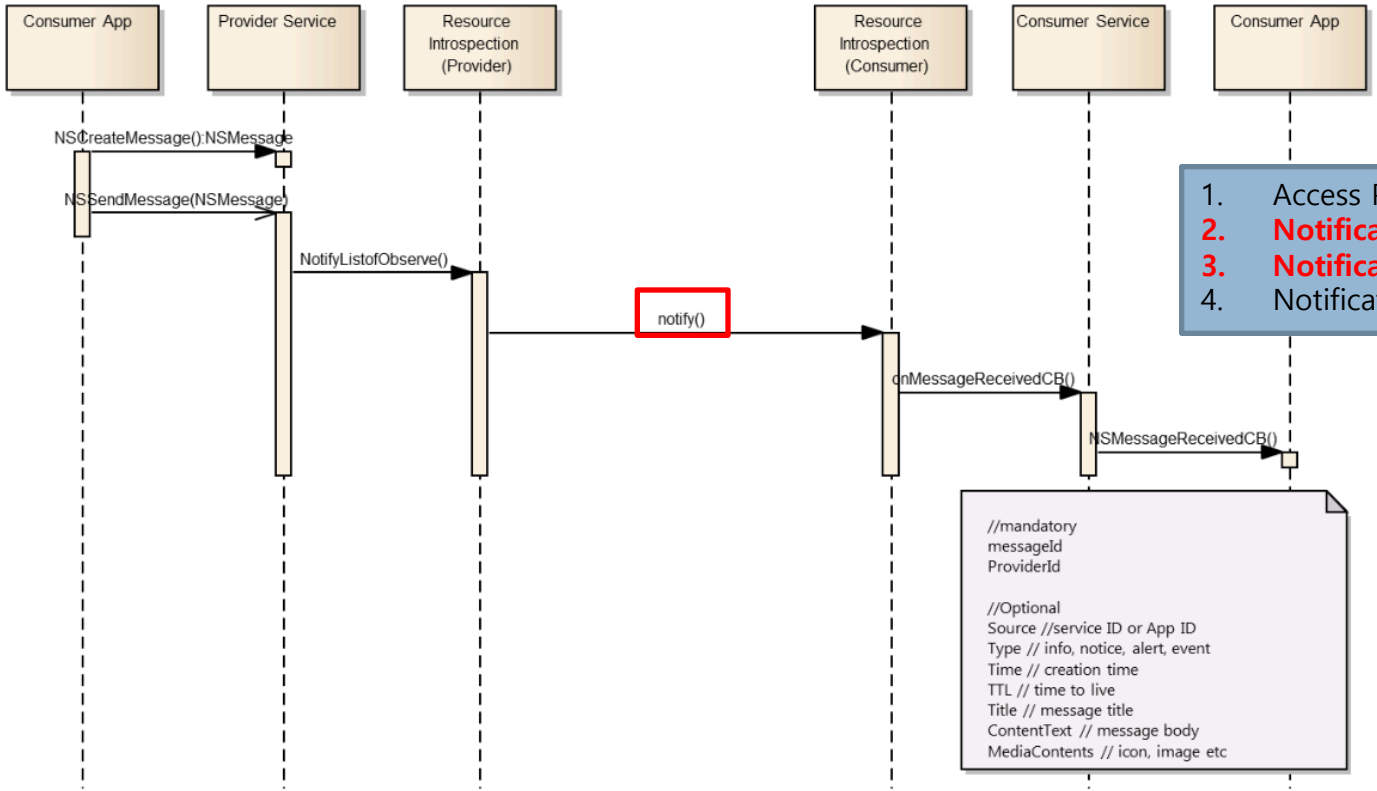
- **Why the subscription message contains 3 distinct URI?**
  - Notification Message : GET (CRUDN)
  - Notification Sync : PUT (CRUDN)
  - Notification Topic : PUT (CRUDN)



Subscription to Notification Resource



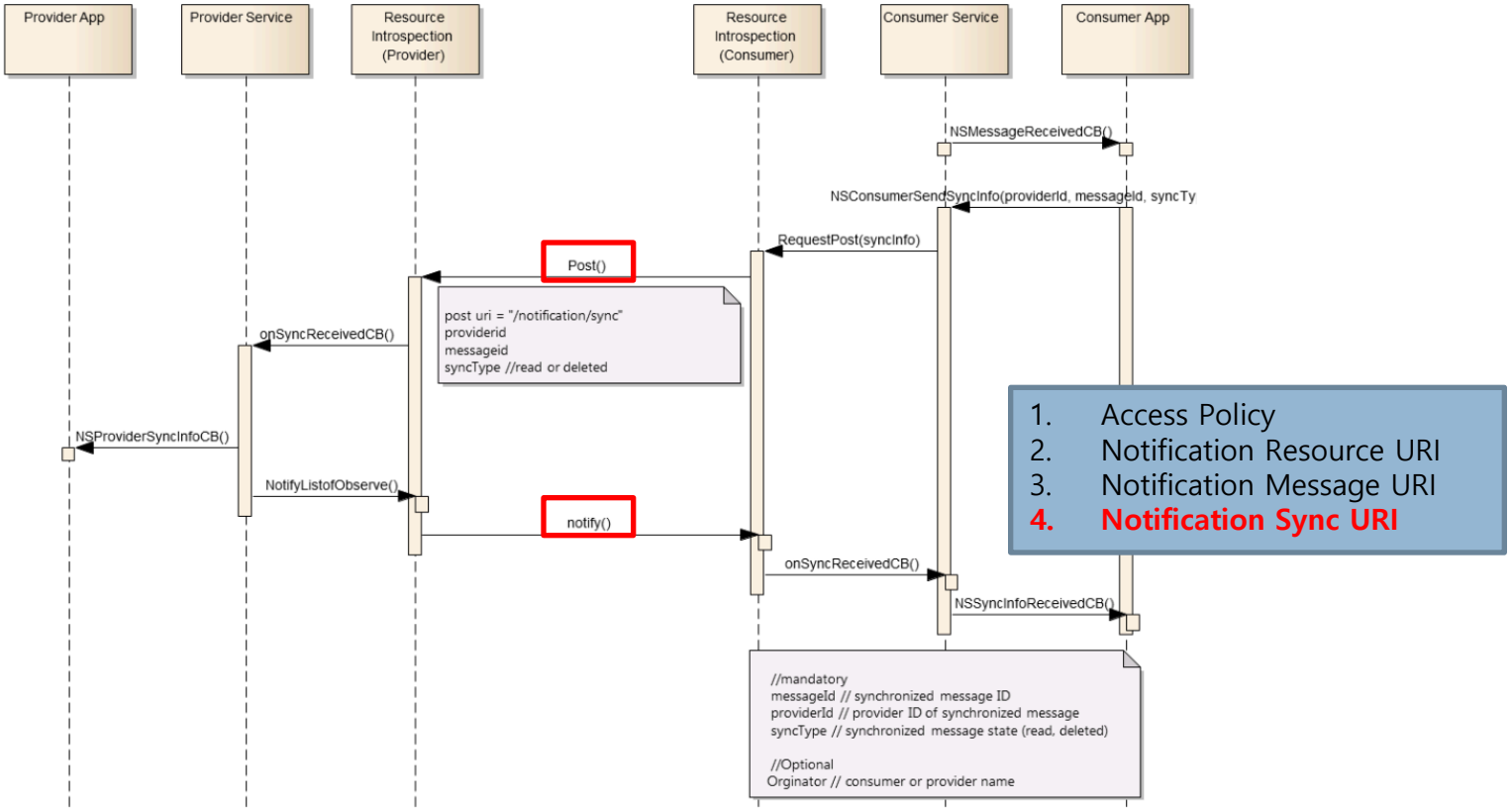
Sending Notify



1. Access Policy
2. **Notification Resource URI**
3. **Notification Message URI**
4. Notification Sync URI

Synchronization of Notification Message

- **Sometimes consumer wants to read the resource synchronously**



Start & Discovery Code Sample


```
case 1:
{
    printf("NSStartProvider(Accepter: Provider)");
    NSProviderConfig config;
    config.subControllability = true;
    config.subRequestCallback = subscribeRequestCallback;
    config.syncInfoCallback = syncCallback;
    config.userInfo = OICStrdup("OCF_NOTIFICATION");
    NSStartProvider(config);
}
    break;

case 2:
{
    printf("NSStartProvider(Accepter: Consumer)");
    NSProviderConfig config;
    config.subControllability = false;
    config.subRequestCallback = subscribeRequestCallback;
    config.syncInfoCallback = syncCallback;
    config.userInfo = OICStrdup("OCF_NOTIFICATION");
    NSStartProvider(config);
}
    break;
```

Access_check

Access_all

<provider>

```
NSConsumerConfig cfg;
cfg.changedCb = onProviderChanged;
cfg.messageCb = onNotificationPosted;
cfg.syncInfoCb = onNotificationSync;
```

```
    printf("1. Start Consumer\n");
    NSStartConsumer(cfg);
    break;
```

<Consumer>

First Notify Message Sending


```
printf("NSSendNotification()");
char title[100];
char body[100];
char topic[100];

printf("id : %d\n", ++id);
printf("title : ");
gets(title);

printf("body : ");
gets(body);

printf("topic : ");
gets(topic);

printf("app - mTitle : %s \n", title);
printf("app - mContentText : %s \n", body);
printf("app - topic : %s \n", topic);

NSMessage * msg = NSCreateMessage();

msg->title = OICStrdup(title);
msg->contentText = OICStrdup(body);
msg->sourceName = OICStrdup("OCF");

if(topic[0] != '\0')
{
    msg->topic = OICStrdup(topic);
}

NSSendMessage(msg);
```

① ② Notify

<provider>

1. Access Policy
2. Notification Resource URI
3. **Notification Message(Topic) URI**
4. Notification Sync URI

```
void onNotificationPosted(NSMessage * notification)
{
    printf("id : %lld\n", (long long int)notification->messageId);
    printf("title : %s\n", notification->title);
    printf("content : %s\n", notification->contentText);
    printf("source : %s\n", notification->sourceName);
    if (notification->topic && strlen(notification->topic) > 0)
    {
        printf("topic : %s\n", notification->topic);
    }
    NSConsumerSendSyncInfo(notification->providerId, notification->messageId, NS_SYNC_READ);
}
```

③

```
case 4:
    printf("4. Select Topics\n");

    if (g_provider && g_topicLL)
    {
        NSTopicLL * iter = g_topicLL;
        int i = 0;
        while (iter)
        {
            iter->state = (i++)%2;
            printf("Topic Name: %s\t Topic State: %d\n", iter->topicName, iter->state);
            iter = iter->next;
        }
        NSConsumerUpdateTopicList(g_provider->providerId, g_topicLL);
    }
    break;
```

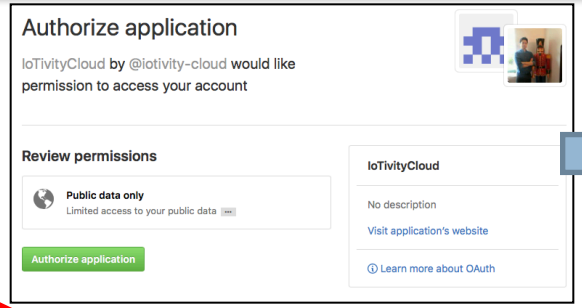
④ Update Topic List and Subscription

<Consumer>

IoTivity Cloud Connection - Provider


```
case 31:
    printf("Remote Server Address: ");
    gets(REMOTE_SERVER_ADDRESS);

    printf("Auth Provider(eg. github): ");
    gets(AUTH_PROVIDER);

    printf("Auth Code: ");
    gets(AUTH_CODE);

1   OCCloudSignup(REMOTE_SERVER_ADDRESS, OCGetServerInstanceIDString(),
        AUTH_PROVIDER, AUTH_CODE, CloudSignupCallback);
    printf("OCCloudSignup requested");
    break;
```

Github Auth code



Auth Code String

```
    printf("Remote Server Address: ");
    gets(REMOTE_SERVER_ADDRESS);

    printf("UID: ");
    gets(UID);

    printf("ACCESS_TOKEN: ");
    gets(ACCESS_TOKEN);

2   OCCloudLogin(REMOTE_SERVER_ADDRESS, UID, OCGetServerInstanceIDString(),
        ACCESS_TOKEN, CloudLoginoutCallback);
    printf("OCCloudLogin requested");
    break;
```

Request Access_Token to

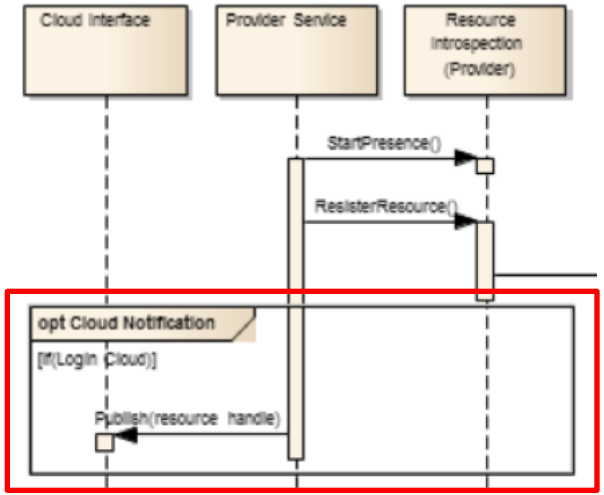
Github send Access_Token to Application

Using Access_Token the Application can access the server

IoTivity Cloud Notification Service Register - Provider


```
printf("Enable Remote Service");
if(!IsCloudLoggedin())
{
    printf("Login required");
    break;
}
NSProviderEnableRemoteService(REMOTE_SERVER_ADDRESS);
break;
```

① <Resource Handle Register>




```
NSPushQueue(DISCOVERY_SCHEDULER, TASK_PUBLISH_RESOURCE, serverAddress);
```

```
case DISCOVERY_SCHEDULER:
{
    NS_LOG(DEBUG, "CASE DISCOVERY_SCHEDULER :");
    pthread_create(&NSThread[i], NULL, NSDiscoverySchedule, NULL);
}
```

<Discovery Scheduler>

```
case TASK_PUBLISH_RESOURCE:
    NS_LOG(DEBUG, "CASE TASK_PUBLISH_RESOURCE : ");
    NSPublishResourceToCloud((char*)node->taskData);
    break;
```

```
OCResourceHandle resourceHandles[1] = {NotificationResource.handle};
OCStackResult res = OCRDPublish(serverAddress, CT_ADAPTER_TCP, resourceHandles, 1,
        &cbData, OC_LOW_QOS);
```

<Notification Resource Handler Register>

IoTivity Cloud Notification Service Register - Consumer