**OCspark/Sparcle Integration Notes**

**Overview**

**The first order of business is to swap out the bug ridden TCP layer of the Sparcle. To that goal there are several phases of incorporating the OCspark:**

1. **OCspark TcpClient Transport Channel**
2. **OCspark RPC / Service Interface**
3. **OcSpark Control/Data channel separation**

**The Transport Channel (1) and RPC (2) are high priority and are the initial focus. The Control/data is per Burcak secondary.**

**Implementation Outline**

1. **Package the OCspark-TcpClient Transport Channel and RPC :**
   * **Configuration file driven**
   * **Unit tests and example programs**
   * **Java Client**

**Status: *Done***

1. **Identify integration points with Sparcle**
   * **How/where in codebase are control and data messages constructed and transmitted**
   * **What is the existing RPC mechanism**

**Status: *Progress made* : see below for my discoveries**

1. **Swap in the OCspark Transport and RPC In place of Sparcle RPC/transport**
   * **Concern: There are no example programs or unit tests to verify correctness**
   * **Concern: What is the build process? The maven pom.xml does not work**

**Status: Issues identified and communicated to Burcak**

**Status: Actual integration work will commence after (2) completed and the issues from (3) explained/addressed**

**Integration Details**

**Following is a “stab” at the Transport and RPC components/layers of Sparcle .**

1. **Java Interface defining the RPC**
   1. **Control API’s look relatively uncomplicated to migrate**
   2. **Data transfer API’s will need to be re-worked**

**Sparcle RPC example: AppContainerService**

**Sparcle Trasport example:**

* **High Level Sparcle RPC Code:** 
  + **Sparcle Java Interface Example: AppContainerService**
* **Low Level Code for Receiving Data**
  + **readCompleted – Code for Receiving Data: reads data on ByteBuffer channel then extracts fields one by one**
  + **Semantics/meaning of fields are hard-coded**
* **Low Level Code for Parsing Received Data: Execute Command**

**High Level Sparcle RPC Code:**

**Sparcle Java Interface Example: AppContainerService**

**Following**

**package** com.pointR.sparcle.io.appsdk;  
  
*/\*\* Container service manages the life cycle of the app and   
exposes communication interface from Sparcle I/O bus  
Its a facade layer sitting on top of Sparcle client  
\*/***public interface** AppContainerService{  
   
 */\*\* SparcleContext to be able to callback the   
 SparcleClient Service \*/* **public void** setSparcleContext(SparcleContext sc);  
   
 */\*\* executes the command send from Application Manager through sparcle,* ***@return*** *Enum* ***@param String*** *[]  
 \*/* **public** Enum excuteAppCommand( String Command );  
   
 */\*\*  
 checks application status and returns an Enum values, indicating the status* ***@return*** *Enum* ***@param String*** *\*/* **public** Enum getAppStatus(String GPURefID );  
 */\*\*  
 Read the next dataset from Sparcle bus to the app.* ***@return*** *byte[]* ***@param String*** *GPU reference ID  
 \*/* **public** ByteBuffer readNextfromSparcle(String GPURefID);  
 */\*\*  
 Writes data into Sparcle bus* ***@return  
 @param String*** *String GPURefID, Byte[] output  
 \*/* **public void** writeAppDatatoSparcle(String GPURefID, ByteBuffer output);  
 **public void** writeFeedDatatoSparcle(String GPURefID, **byte**[] output);  
 **public void** writeAppNativeDatatoSparcle(String GPURefID, NativeBuffer nBuffer);  
  
*/\* \*//\*\*  
 wrapper API that writes the processed output into sparcle bus and reads the next dataset for processing  
 \*//\*  
 public ByteBuffer appPush (String GPURefID , ByteBuffer payload );  
  
\*/* **public** String excuteAppCommand(String Command, String GPURefID);  
   
 **public** AppSdkService getAppSdkServiceInstance();  
  
 */\*\*  
 \* Method to find the app is running or not for the given GpuRefId(APP id)  
 \*   
 \** ***@param gpuRefId*** *\** ***@return*** *\*/* **boolean** isAppRunning(String gpuRefId);  
}

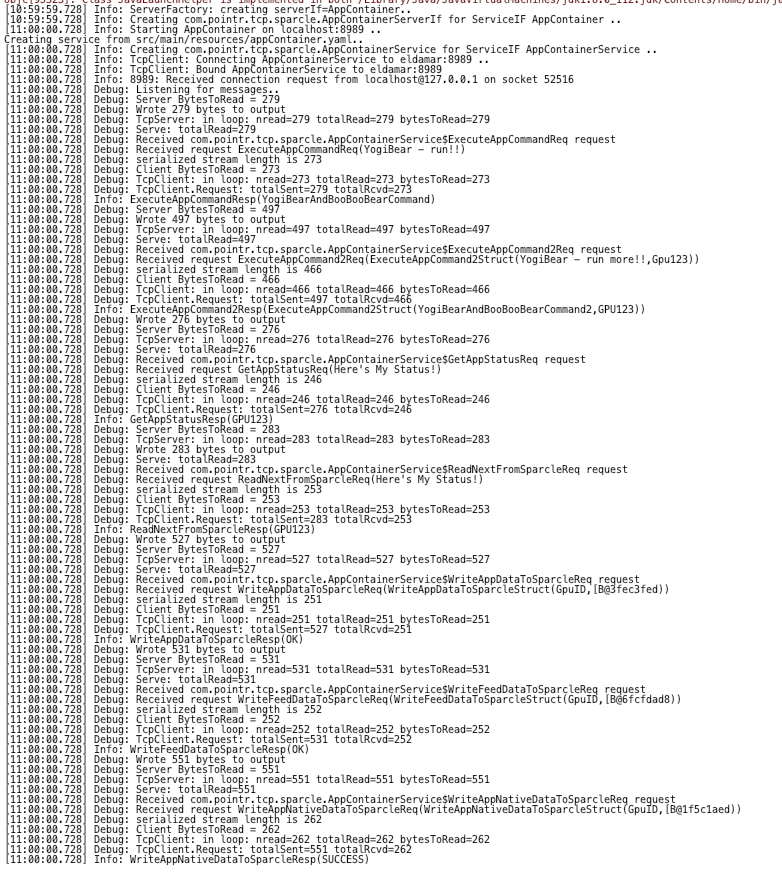
**Low Level Code for Receiving Data**

**public void** readCompleted(ByteBuffer buffer) {  
 buffer.order(ByteOrder.***LITTLE\_ENDIAN***);  
 buffer.flip(); *// enabling received buffer to read mode  
 //append received buffer to filebuffer* **try** {  
 **fileBuffer**.put(buffer);  
 }  
 **catch**(Exception e) {  
 e.printStackTrace();  
 ***log***.error(**"{},{},{},SC Fail to extract metadata buffer overflow,{}"**, System.*currentTimeMillis*(), SparcleConstants.***LOG\_SME***, **appContext**.getGPURefId()  
 );  
 **fileBuffer**.clear();  
 **fileBuffer**.position(0);  
 **fileBuffer**.flip();  
  
 **firstBuffer** = **true**;  
 **fileSize**=SparcleConstants.*MAX\_FILE\_SIZE*;  
 **metaData** = **new** HashMap<>();  
 }  
  
  
 *//if it is first buffer get data meta data from the buffer* **if**(**firstBuffer** && **fileBuffer**.position() >= SparcleConstants.***META\_DATA\_SIZE***) {  
 **fileBuffer**.flip(); *//enabling file buffer to read mode  
   
 //read meta data from the file buffer* **firstBuffer**= **false**;  
  
 *//do actions on meta data* extractMetaData(**fileBuffer**);  
   
   
 *//remove meta data from the file buffer  
 //fileBuffer.compact();* **int** pos = **fileBuffer**.limit();  
 **fileBuffer**.clear();*//enabling write mode* **fileBuffer**.position(pos);  
 ***log***.debug(**"{},{},{},{}"**,System.*currentTimeMillis*(), SparcleConstants.***LOG\_SCRS***,  
 **appContext**.getGPURefId(), **metaData**.get(MetaDataEnum.***name***.toString()));  
 }  
   
   
 *//save received file into cache* **if** (**fileBuffer**.position() >= **fileSize**) {  
 **if**(SparcleConstants.*USE\_NATIVE\_INTERFACE*) {  
 ***log***.info(**"{},{},{},{},{}"**,System.*currentTimeMillis*(), SparcleConstants.***LOG\_SCRC***,  
 **appContext**.getGPURefId(), **metaData**.get(MetaDataEnum.***name***.toString()),  
 **sparcleClient**.getNativeCacheInstance().getCount(**appContext**.getGPURefId(), SparcleCacheTypes.***iCache***));   
 } **else** {  
 ***log***.info(**"{},{},{},{},{}"**,System.*currentTimeMillis*(), SparcleConstants.***LOG\_SCRC***,  
 **appContext**.getGPURefId(), **metaData**.get(MetaDataEnum.***name***.toString()),  
 **sparcleClient**.getCacheInstance().getCount(**appContext**.getGPURefId(), SparcleCacheTypes.***iCache***));   
 }  
  
 **fileBuffer**.flip();   
 **fileBuffer**.get(**currentFile**);  
   
 *//removing the current file from file buffer,* **fileBuffer**.compact();  
   
 *//for file* **if**(**metaData**.get(MetaDataEnum.***type***.toString()).equals(MetaDataTypesEnum.***FILE***.toString())) {  
 putReceivedPayloadToCache();  
 }  
   
 *//for app info* **else if** (**metaData**.get(MetaDataEnum.***type***.toString()).equals(MetaDataTypesEnum.***APP\_INFO***.toString())) {  
 String appInfoString = **new** String(Arrays.*copyOfRange*(**currentFile**, SparcleConstants.***META\_DATA\_SIZE***-1, **currentFile**.**length**), **cs**);  
 appInfoString = appInfoString.trim();  
 **if**(appInfoString.length()>0) {  
 HashMap<String, String> appInfo = extractAppInfo(appInfoString);  
 **if**(appInfo.size() >= 4) {  
 **sparcleClient**.updateRefID(**appContext**.getGPURefId(), appInfo.get(AppInfoEnum.***GPU\_REF\_ID***.toString()));  
 **appContext**.setClientConfig(**new** SparcleClientConfig(appInfo.get(AppInfoEnum.***GPU\_REF\_ID***.toString()),  
 appInfo.get(AppInfoEnum.***GPU\_ID***.toString()), appInfo.get(AppInfoEnum.***APP\_NAME***.toString()),  
 appInfo.get(AppInfoEnum.***APP\_INFO***.toString()), **null**, **null**, **null**));  
 **sparcleClient**.getAppInfo(**appContext**, appInfo.get(AppInfoEnum.***GPU\_REF\_ID***.toString()));  
 **if**(appInfo.get(AppInfoEnum.***APP\_NAME***.toString()).equals(SparcleConstants.***FEEDING\_APP\_NAME***)) {  
 **sparcleClient**.updateSCWriteListener(**appContext**);   
 }  
  
 } **else if**(appInfo.size() >= 1) {  
 **sparcleClient**.updateRefID(**appContext**.getGPURefId(), appInfo.get(AppInfoEnum.***GPU\_REF\_ID***.toString()));  
 **sparcleClient**.getAppInfo(**appContext**, appInfo.get(AppInfoEnum.***GPU\_REF\_ID***.toString()));  
 }  
 }  
 }  
 **else if** (**metaData**.get(MetaDataEnum.***type***.toString()).equals(MetaDataTypesEnum.***COMMAND***.toString())) {  
 **sparcleClient**.executeAmCmd(**appContext**, **metaData**, **currentFile**);  
 } **else** {  
 ***log***.error(**"{},{},{},Unknown buffer header: {}"**, System.*currentTimeMillis*(), SparcleConstants.***LOG\_ERROR***,  
 **appContext**.getGPURefId(), **new** String(**metaDataBuffer**, **cs**));  
   
 }  
   
 *//reset first buffer* **firstBuffer** = **true**;  
 **this**.**fileSize** = SparcleConstants.*MAX\_FILE\_SIZE*;  
 **if**(**fileBuffer**.position() > SparcleConstants.***META\_DATA\_SIZE***) {  
 buffer.clear();  
 readCompleted(buffer);  
 }  
 }  
   
 buffer.clear();  
}

**Low Level Code for Parsing Received Data: Execute Command**

**public void** executeAmCmd(AppContext appContext, HashMap<String, String> metaData, **byte**[] currentFile) {  
  
 String commandName = metaData.get(MetaDataEnum.***name***.toString());  
 String command = **new** String(  
 Arrays.*copyOfRange*(currentFile, SparcleConstants.***META\_DATA\_SIZE*** - 1, currentFile.**length**), **cs**).trim();  
 **if** (commandName.equals(SparcleAppCommands.***APP\_START***.toString())  
 || commandName.equals(SparcleAppCommands.***APP\_UPDATE***.toString())  
 || commandName.equals(SparcleAppCommands.***APP\_REVERT***.toString())  
 || commandName.equals(SparcleAppCommands.***START\_CAMERA\_STREAM***.toString())  
 || commandName.equals(SparcleAppCommands.***STOP\_CAMERA\_STREAM***.toString())) {  
 commandName = commandName + SparcleConstants.***APP\_INFO\_DELIMITER*** + command;  
 }  
 ***log***.debug(**"{} SC:Read received {} Command for GpuRefId: {}"**, System.*currentTimeMillis*(), commandName,  
 appContext.getGPURefId());  
 String isExecuted;  
  
 **if** (commandName.contains(SparcleAppCommands.***APP\_START***.toString())  
 || SparcleAppCommands.***APP\_STOP***.toString().equals(commandName)  
 || commandName.contains(SparcleAppCommands.***APP\_UPDATE***.toString())  
 || commandName.contains(SparcleAppCommands.***APP\_REVERT***.toString())  
 || commandName.contains(SparcleAppCommands.***START\_CAMERA\_STREAM***.toString())  
 || commandName.contains(SparcleAppCommands.***STOP\_CAMERA\_STREAM***.toString())  
 || SparcleAppCommands.***CLEAN\_CACHE***.toString().equals(commandName)) {  
 *// Call Sparcle client method to start\_app.* System.***out***.println(**"commandName: "**+commandName);  
 isExecuted = *sparcleClient*.executeAppCommand(commandName, appContext).toString();  
 ByteBuffer payload = BufferPool.*poll*(appContext.getGPURefId(), SparcleCacheTypes.***oCache***.toString());  
 payload.put(**metaDataService**.getMetaData(MetaDataTypesEnum.***COMMAND***,  
 metaData.get(MetaDataEnum.***name***.toString()), isExecuted.length()));  
 payload.put(ByteBuffer.*wrap*(isExecuted.getBytes()));  
 payload.flip();  
  
 appContext.setAppCacheServiceable(**false**);  
 Connection connection = **connectionFactory**.getConnection(appContext.getGPURefId());  
 **while** (!connection.getSparcleWriteListener().isWriteCompleted()) {  
 **try** {  
 Thread.*sleep*(2);  
 } **catch** (InterruptedException e) {  
 *//* ***TODO Auto-generated catch block*** e.printStackTrace();  
 }  
 }  
 *sparcleClient*.writeToSparcle(appContext, payload);  
 appContext.setAppCacheServiceable(**true**);  
 *sparcleClient*.writeToSparcle(appContext, **null**);  
  
 } **else if** (SparcleAppCommands.***APP\_STATUS***.toString().equals(command)) {  
  
 String response = *sparcleClient*.getAppStatus(appContext.getGPURefId()).toString();  
 ByteBuffer payload = BufferPool.*poll*(appContext.getGPURefId(), SparcleCacheTypes.***oCache***.toString());  
 payload.put(**metaDataService**.getMetaData(MetaDataTypesEnum.***COMMAND***, command, response.length()));  
 payload.put(ByteBuffer.*wrap*(response.getBytes()));  
 payload.flip();  
  
 appContext.setAppCacheServiceable(**false**);  
 Connection connection = **connectionFactory**.getConnection(appContext.getGPURefId());  
 **while** (!connection.getSparcleWriteListener().isWriteCompleted()) {  
 **try** {  
 Thread.*sleep*(2);  
 } **catch** (InterruptedException e) {  
 *//* ***TODO Auto-generated catch block*** e.printStackTrace();  
 }  
 }  
 *sparcleClient*.writeToSparcle(appContext, payload);  
 appContext.setAppCacheServiceable(**true**);  
 *sparcleClient*.writeToSparcle(appContext, **null**);  
 } **else if** (SparcleAppCommands.***RELOAD***.toString().equals(commandName)) {  
  
 System.***out***.println(**" reset command call in execute app command in client "**);  
 String response = **""**;  
 ByteBuffer payload = BufferPool.*poll*(appContext.getGPURefId(), SparcleCacheTypes.***oCache***.toString());  
 payload.put(**metaDataService**.getMetaData(MetaDataTypesEnum.***COMMAND***,  
 metaData.get(MetaDataEnum.***name***.toString()), response.length()));  
*// payload.put(ByteBuffer.wrap(response.getBytes()));* **if** (!response.equals(**""**)) {  
 payload.put(ByteBuffer.*wrap*(response.getBytes()));  
 }  
 payload.flip();  
  
 appContext.setAppCacheServiceable(**false**);  
 Connection connection = **connectionFactory**.getConnection(appContext.getGPURefId());  
 **while** (!connection.getSparcleWriteListener().isWriteCompleted()) {  
 **try** {  
 Thread.*sleep*(2);  
 } **catch** (InterruptedException e) {  
 *//* ***TODO Auto-generated catch block*** e.printStackTrace();  
 }  
 }  
 **try** {  
 String res = *sparcleClient*.reload(appContext);  
 }  
 **catch**(Exception e) {  
 ***log***.error(**"{},{},{},reason:{}"**, System.*currentTimeMillis*(),SparcleConstants.***LOG\_ARELOADF***, appContext.getGPURefId(),e.getMessage());  
 }  
 *sparcleClient*.writeToSparcle(appContext, payload);  
 } **else** {  
 ***log***.error(**"{},{},{},Issues in processing command : {}"**, System.*currentTimeMillis*(),  
 SparcleConstants.***LOG\_ACF***, appContext.getGPURefId(),**new** String(currentFile).toString());  
 }  
 }

**Results**

****