Use Case No 27: Data Acquisition

1. Summary:

This document uses an analysis of the State Estimation use case to propose a set of IDL interface classes for a Data Acquisition component that can be used in other, yet to be defined, use cases.

Since Measurements are a very generic entity, this interface specification may cover a wide range of use cases that involve the transfer of telemetered data from a Data Acquisition application. It is likely to be equally applicable to Automatic Generation Control or updating user interfaces. The main difference between the client applications is in the way they select which measurements are of interest. The other difference is that there is likely to be only one State Estimator in the system but there will be many user interfaces.

Within this document CIM_ is used as a prefix for entities within a component as if they were directly implemented using the Common Information Model (CIM) as a schema. CIS_ is used as a prefix for Component Interface Specification structures and classes.

2. Information Exchanges between the State Estimator & Data Acquisition

There are the following exchanges between Data Acquisition and Topology Processor and between Data Acquisition and State Estimator. The Topology Processor consumes switch status measurements, whilst the State Estimator consumes and produces analogue values and quality.

The size of the information exchanges is based on a 1000 bus system but is intended for comparative purposes only. The time scale of the information exchanges is based on allowing dynamic transients on a power system to settle after a switching operation.

ID	Producer Actor or System	Consumer Actor or System	Exchange Type	Information Content
F	Data Acquisition	Topology Processor	Fast Data Event	Individual breaker status change event (initiated by Data Acquisition) 0-10 per second (max)
G	Data Acquisition	Topology Processor	Array Data Set	Full network ConductingEquipment states (initiated by Topology Processor) 10000 every 3 minutes (max)
I	Data Acquisition	State Estimator	Array Data Set	Relevant analogue measurement values and quality attributes (initiated by State Estimator) 6000 every 5 minutes (typical)
K	State Estimator	Data Acquisition	Array Data Set	Estimated measurement values and quality 10000 per 5 minutes (typical)

The event trace diagram requires more detail about what information is exchanged and in which order.

It is assumed that each component has an internal representation of the set of CIM_Measurements (kV, MW, MVAR etc.) for the network. Each component will have its own internal representation of several CIM_MeasurementValues for each Measurement. These may be actually implemented as multiple attributes within a measurement record.

For example:

Data Acquisition holds internally <set of telemetered measurement values>, <set of manually replaced values>, <set of estimator calculated values>. For this use case it must provide <set of current values> regardless of the source.

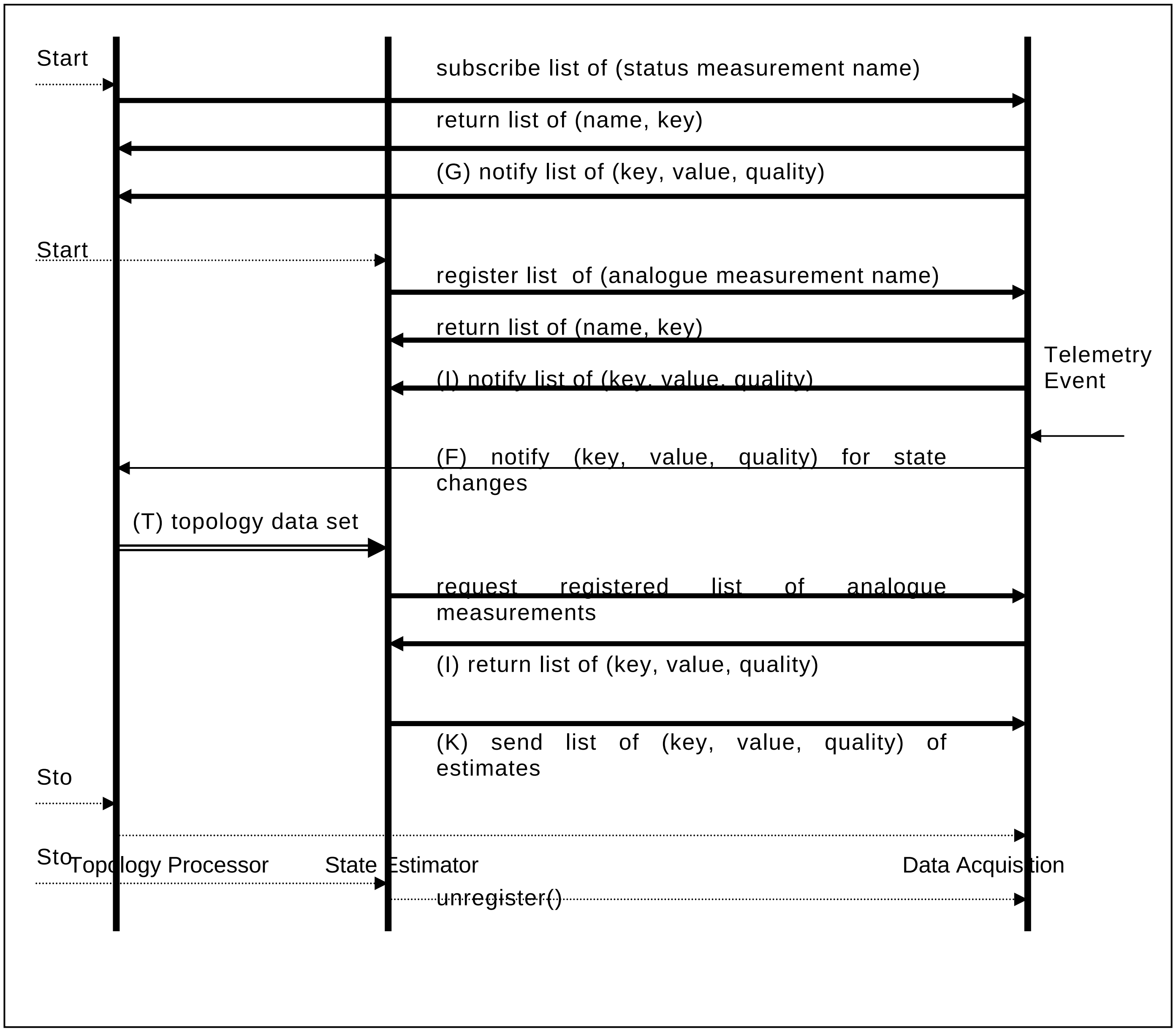
State Estimator will have <set of input measurements>, <set of output measurements>.

3. Information exchange packaging.

There are several ways to define which particular instances are involved in each exchange. Some of these have been used in the event trace diagram.

- **One-way send**. Producer application has implicit or internal knowledge of what consuming applications require. E.g. State Estimator produces array of estimated measurement values for Data Acquisition. This is the same as Subscribe & Notify with the Subscribe stage implicitly defined in the producer application.
- **Request & Reply**. Producer application sends data to consumer applications, based on consumer application requesting data. E.g. State Estimator requests array of analogue measurements. This is essentially the same as a one-off Subscribe & Notify with immediate response.
- **Subscribe & Notify Changes**. Producer application sends data changed since previous request to consumer application. The producer application must maintain a list for each consumer application to identify which data items have changed since the last successful data exchange. When data changes the producer application must scan its interest lists to establish whether any data needs sending. The initial subscription would be followed by an immediate reply of all the subscribed data to synchronise the applications.
- **Broadcast & Filter**. Middleware supports producer application broadcasting events and consumer applications set up filter profiles. Unless there are multiple consumers of many events, this is less efficient of the infrastructure bandwidth than the Subscribe & Notify.

4. Event trace diagram


Start
subscribe list of (status measurement name)
return list of (name, key)
(G) notify list of (key, value, quality)
Start
register list of (analogue measurement name)
return list of (name, key)
(I) notify list of (key, value, quality)
Telemetry Event
(F) notify (key, value, quality) for state changes
(T) topology data set
request registered list of analogue measurements
(I) return list of (key, value, quality)
(K) send list of (key, value, quality) of estimates
Sto
Sto
Topology Processor
State Estimator
Data Acquisition
unregister()

5. Interface structures - what is exchanged

For each type of application and type of exchange, it is necessary to define *what* entities and attributes are exchanged. This means defining *interface structures* to hold the data. These structures may be subsets or views of CIM entities. In other words the CIM is used as a data dictionary. Depending on the technology, the interface structures may be compiled or interpreted at run time.

In this particular use case the specific attributes to be exchanged are defined at compile time within the interface specification.

5.1 Single status data event (exchange F):

For performance reasons, these fast events would use specific interface structures that are specialised views of CIM entities. The following structure is appropriate.

```
CIS_MeasurementValue
{
        key             = CIM Measurement.key
        value           = CIM Measurement->MeasurementValue.value
        quality         = CIM Measurement->MeasurementValue.quality
}
```

The quality status word must include as a minimum:

Invalid	yes/no

In this example it is assumed that the key will be numeric for efficiency reasons. This is discussed in a later section.

5.2 Array of status data (exchange G):

This is essentially a generalisation of single status event described above and hence should use an array of the structures described above.

5.3 Array of telemetered analogue data (exchange I):

This is essentially the same as exchange type G above but using a different set of measurements. It can therefore use the same array of the structures described above with the addition of the quality bit:

EstimatorReplaced	yes/no

This information allows a State Estimator to use a different weight for telemetered measurements and estimated measurements. It does however imply that the Data Acquisition component has built-in knowledge that the State Estimator is a potential source of data.

5.4 Array of estimated analogue data (exchange K):

This is identical to exchange type I but with different producer and consumer applications. However there may be different quality flags. Possible flags include:

Invalid	yes/no,	state estimator has valid solution
OverRange	yes/no,	value exceeds limits

Suspect	yes/no, state estimator declared bad telemetered MeasurementValue for this Measurement.

The Suspect flag indicates that the estimated MeasurementValue has been significantly different from the telemetered MeasurementValue for a number of State Estimator solutions. It does not mean that the estimated values sent to the Data Acquisition component are suspect. In other words, this quality bit applies to the Measurement entity as a whole not an individual MeasurementValue.

6. IEC 61850-7-4 Quality bits.

For reference, the following quality flags are defined for substation automation.

Quality bit	Description
BadReference	Measurement value may be incorrect due to a reference being out of calibration.
CommFailure	Measurement value is not valid due to a communication failure.
Blocked	Measurement value is blocked (unavailable) for transmission.
Substituted	Measurement value has been substituted, e.g. by input of an operator, or software.
NotTopical	Measurement value is old and possibly invalid, as it has not been successfully updated during a specified time interval.
Invalid	Measurement value may be incorrect and should not be used. *It is not clear whether this is an OR of other quality bits)*
OverFlow	Measurement value is beyond the capability of being represented properly. For example, a counter value overflows from maximum count back to a value of zero.
OverRange	Measurement value is beyond a predefined range of value.
TransientState	Measurement value is due to a transient condition.
Test	Measurement value is transmitted for test purposes.
DefaultValue	Measurement value is a default value.

For this use case, EstimatorReplaced and Suspect are also required.

EstimatorReplaced	Measurement is an estimated value. A State Estimator may use a different weight for telemetered measurements and estimated measurements.
Suspect	The estimated measurement value has been significantly different from the telemetered value for a number of State Estimator solutions.

7. Identification - which instances are exchanged

The structure defined above used a key for measurements that is unique within the system. The event trace diagram shows an information exchange that allows the State Estimator to obtain these keys from the Data Acquisition component. An alternative is for both components to get these keys from the data definition system or a shared naming service.

A suitable structure is:

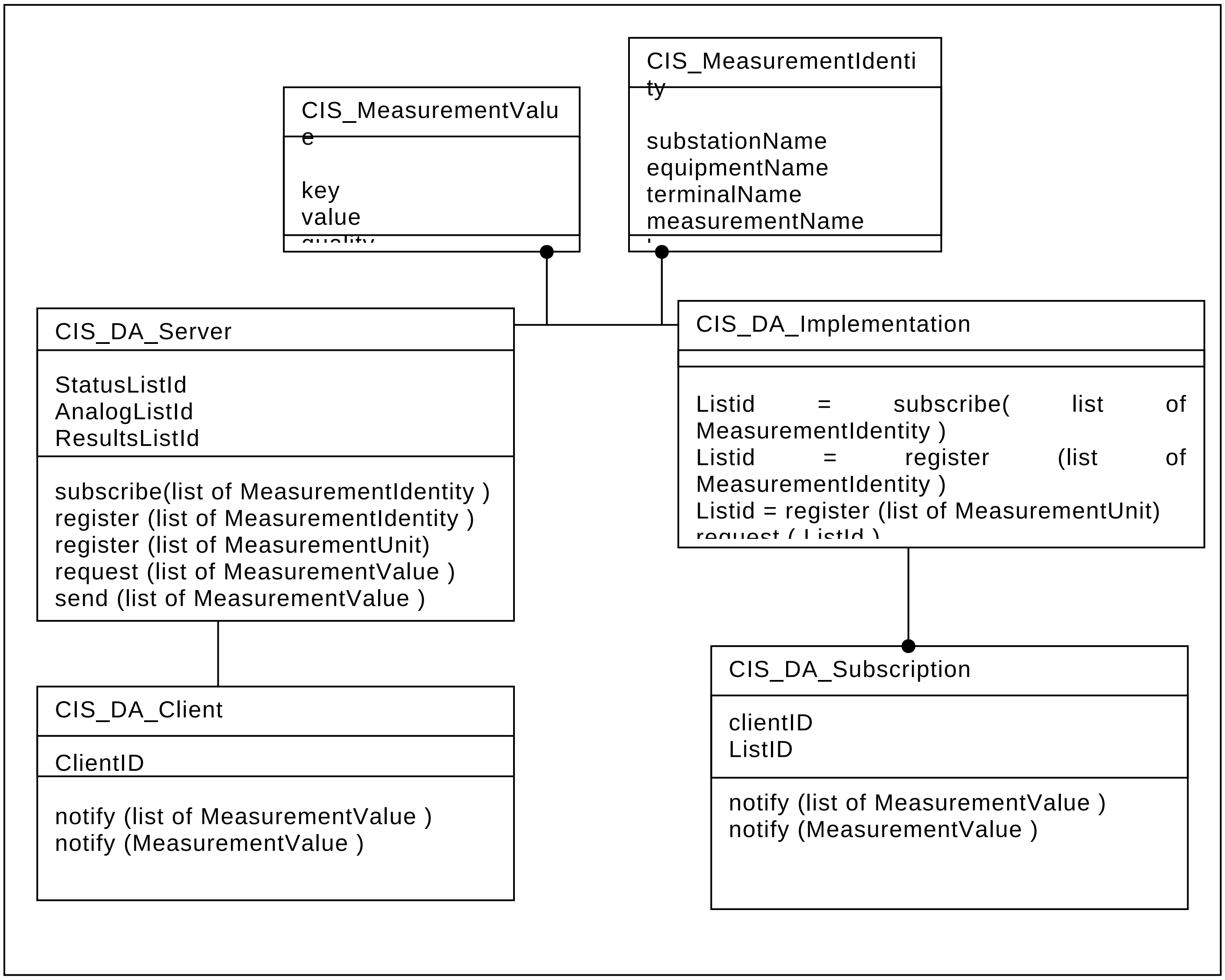
```
CIS_MeasurementIdentity
{
        substationName,               //unique within control centre
        conductingEquipmentName, //unique within Substation
        terminalNumber,               //unique within ConductingEquipment
        measurementName,            //unique for each Terminal
        key                                  //unique within control centre
}
```

The State Estimator could then send an array of these structures with identifier =0 to either a database application or Data Acquisition itself. This application would reply with the same structure with non-zero identifiers for valid measurements.

The set of measurements to be exchanged can be defined in several different ways. A standard interface should allow any of the following.

a. Common Database. E.g. Data Acquisition & State Estimator are initialised from the same database. Measurements are marked in the database as used by the State Estimator. The Data Acquisition can set up registration structures on start up without requiring a information exchange of CIS_MeasurementIdentity.
b. Coded in Data Acquisition. Data Acquisition implicitly knows that State Estimator is interested in particular MeasurementUnits for all ConductingEquipment. In this case the ‘register’ information exchange would have an alternative form.
c. Coded in State Estimator. State Estimator subscribes for measurements that match its data. The simplest subscription is a list of specific measurements. However State Estimator could automatically generate this measurement list based on a list of valid measurement units and a list of Substations and ConductingEquipment loaded in the State Estimator internal model.

8. Interface class diagram


CIS_MeasurementValue
key
value
CIS_MeasurementIdentity
substationName
equipmentName
terminalName
measurementName
CIS_DA_Server
StatusListId
AnalogListId
ResultsListId
subscribe(list of MeasurementIdentity)
register (list of MeasurementIdentity)
register (list of MeasurementUnit)
request (list of MeasurementValue)
send (list of MeasurementValue)
CIS_DA_Implementation
Listid = subscribe(list of MeasurementIdentity)
Listid = register (list of MeasurementIdentity)
Listid = register (list of MeasurementUnit)
CIS_DA_Client
ClientID
notify (list of MeasurementValue)
notify (MeasurementValue)
CIS_DA_Subscription
clientID
ListID
notify (list of MeasurementValue)
notify (MeasurementValue)

CIS_MeasurementValue	interface structure described above
CIS_MeasurementIdentity	interface structure described above
CIS_MeasurementUnit	interface structure (not shown)
CIS_DA_Server	public interface class as used by State Estimator or other client of Data Acquisition.
CIS_DA_Implementation	private interface class implemented within Data Acquisition.
CIS_DA_Subscription	private interface class implemented within Data Acquisition and instantiated for each client.
CIS_DA_Client	public interface class provided by State Estimator and other clients to receive notifications from Data Acquisition.

9. Interface Definition Language example.

The following assumes that long sequences are sent as a set of variable length 'packets'. The concept is that each packet transfers part of an array. The packet therefore contains information on the number of items in the packet and the offset from the start. By passing the offset with each information exchange, there is some protection for packets arriving in the wrong order.

```
for ( offset = 0; offset < bigArrayLength; )
{
n = Send ( bigArray[offset], offset, STDSEQSIZE);
offset += n;
}
```

File: CIS_MeasStruct.idl

```
#ifndef CIS_MeasStruct_idl
#define CIS_MeasStruct_idl

#define STDSEQSIZE 1024

struct CIS_MeasurementValue
{
        long            key;
        float           value;
        unsigned long   quality;
};

struct CIS_MeasurementIdentity
{
        string          substationName;
        string          equipmentName;
        string          terminalName;
        string          measurementName;
                                                long
};

typedef         sequence<CIS_MeasurementValue,          STDSEQSIZE          >
                CIS_MeasValueSeq;
typedef         sequence<CIS_MeasurementValue,          STDSEQSIZE          >
                CIS_MeasValueSeq;

#endif
```

File: CIS_DA_Server.idl

```
#ifndef CIS_DA_Server_idl
#define CIS_DA_Server_idl

#include "CIS_MeasStruct.idl"

interface CIS_DA_Client;

interface CIS_DA_Server
{
/*        subscribe to notifications on selected measurements.
          ClientID is the CORBA identifier of this process
          NumberOfItems is number of items in sequence
          Offset is number of items already received (zero to start)
          Result is number of items returned. Zero means end of list.
*/
          short   subscribe (in CIS_DA_Client clientID,
                            in short offset,
                            in short numberOfItems,
                            inout CIS_MeasIdSeq );

/*        register a list of names and obtain matching keys
          RegisteredListID is reference to list
*/
          short   register (out long RegisteredListID,
                            in short offset,
                            in short numberOfItems,
                            inout CIS_MeasIdSeq seq,);

/*        send a list of measurement values
*/
          void    send (in long RegisteredListID,
                            in short offset,
                            in short numberOfItems,
                            in CIS_MeasValueSeq seq);

/*        request a list of previously registered measurement values
*/
          short   request  (in long RegisteredListID,
                            in short offset,
                            in short numberOfItems,
                            out CIS_MeasValueSeq seq);
};
#endif
```

File: CIS_DA_Client.idl

```
#ifndef CIS_DA_Client_idl
#define CIS_DA_Client_idl

#include "CIS_MeasStruct.idl"

interface CIS_DA_Client
{
/*        receive notification for a single measurement
*/
          oneway void    notifyItem (in CIS_MeasurementValue item);

/*        receive a list of previously registered measurement values
*/
          short    notifySeq  (in long RegisteredListID,
                               in short offset,
                               in short numberOfItems,
                               in CIS_MeasValueSeq seq);
};
#endif
```

References:

This analysis is based on the State Estimation and Network Modification use cases using the methodology described in Information Exchange Design.

Revision History:

No	Date	Author	Description
1.	28-Apr-99	T. Berry	Original
2.	5-May-99	T. Berry	Add IDL
3.	7-May-99	T. Berry	Clarify use of CIM and CIS prefixes.