MICE Configuration DB

Janusz Martyniak, Imperial College London

MICE CM39 Software Paralell

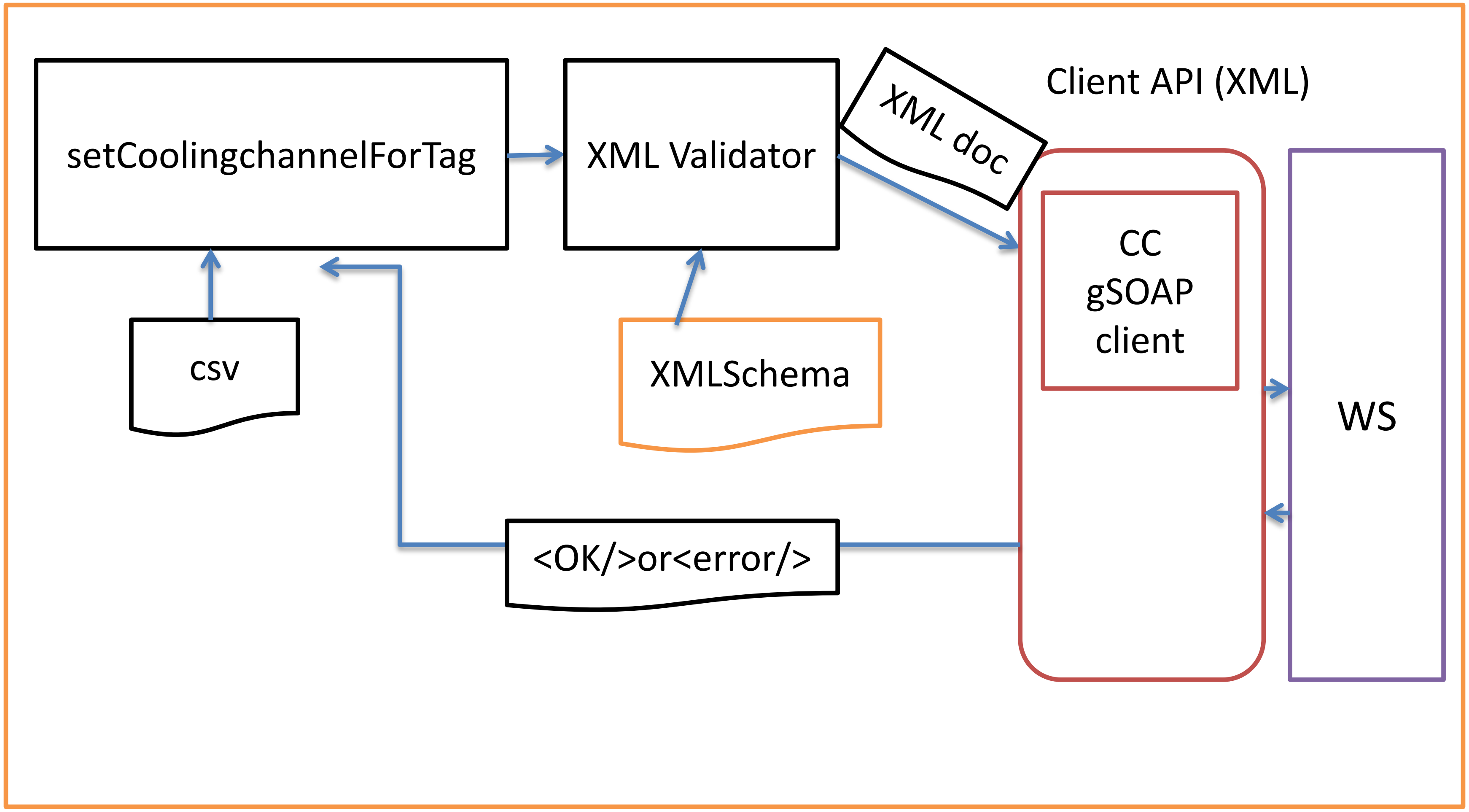
Cooling Channel Client API

- The server code exists (jax-ws on Tomcat)
- Easy to create client bindings in java, Python (preferred), but also in C or C++
- Cooling Channel software itself is written in C, so it was most natural to provide the CDB client API in this language
- Use gSoap toolkit to generate C client-side bindings

Cooling Channel *setX()* Workflow

- To be performed on a supermouse enabled DB (MLCR)
- User creates a csv file with data to be submitted.
- A single API function is called (takes a file handle as an argument)
- csv is converted to an XML document
- XML doc is *validated* against a XML schema
- gSoap client request in made

setCoolingchannelForTag(..)



getCoolingchannelForTag(endp, tag)

- Call a gSOAP client passing an endpoint and a tag
- Get an XML document as a response
- Build a coolingchannel C struct and return it to a user
- User can walk through the struct and retrieve magnet and coil data.

Coolingchannel(s) struct(s)

```
/*
 *  Cooling channel structure. Maintains both direct and sequential access to magnets
 */
 struct coolingchannel {
    char *validfromtime;
    char *validuntiltime;
    CC_NS1_hashtable_t *magnets; /* magnets for this coolingchannel */
    cc_magnet_t *magnet; /* first magnet for this coolingchannel */
    struct coolingchannel *next; /* next coolingchannel */
 };
 typedef struct coolingchannel coolingchannel_t;

 /*
  * Coolingchannels (plural!). Maintains sequential access to
  * cooling channels.
  */
 struct coolingchannels {
    coolingchannel_t *cc; /* first coolingchannel */
 };
 typedef struct coolingchannels coolingchannels_t;
```

Magnets

```
/*
  * Cooling channel magnet structure. Maintains direct access to coils via
  * a hashtable and a sequential access via a linked list
  */
 struct cc_magnet {
    char *name;
    char *mode;
    pol_t polarity;
    CC_NS1_hashtable_t *coils; /* coils for this magnet */
    cc_coil_t *coil; /* first coil for this magnet */
    struct cc_magnet *next; // next magnet, or NULL if the last one (for seq.
access)
 };
 typedef struct cc_magnet cc_magnet_t;
```

Coils

```
/*
 *  Cooling channel coil. Contains a hash table of attributes.
 */
struct cc_coil {
   char *name; /* coil name */
   CC_NS1_hashtable_t *attr; /* attribute hash table */
   struct cc_coil *next; /* next coil (for seq. access) */
};
typedef struct cc_coil cc_coil_t;
```

usage

Looping over structs is easy, i.e. for magnets:

```
 if (coolingchannel != NULL) {
       printf("  Coolingchannel: valid from:%s, valid until: %s\n”,
          coolingchannel->validfromtime, coolingchannel->validuntiltime);
       for (magnet = coolingchannel->magnet; magnet != NULL; magnet
              = magnet->next) {
             …..
        }
}
```

Software status

- get/setCoolingchannelForTag ready
- on launchpad
- Other operations ready on XML level, have to plug-in an XML validation step for set functions.
- Unit tests exist (Cunit)
- Should be ready for tests in 1-2 weeks

Batch Iteration Number API

Create a new table to store a relation between a batch iteration number used by the Grid reconstruction system and MAUS. This will allow to reprocess the data to allow for any changes not related to the MAUS version itself.

The batch reconstruction number will determine the datacards which used by MAUS.

The datacards will be stored in the DB.

See: http://micewww.pp.rl.ac.uk/issues/1285

Batch Iteration Number API

- Relevant table created in the DB (locally)
- WS server side implemented (java)
- WS client API written in Python
- Tested locally
- Further development postponed – priority given to the Coolingchannel C API
- Not tested with MAUS