How to implement pseudo-synchronous (queued) instrumentation with AGENT++?

Motivation

The standard and easy way of instrumenting a MibLeaf object is synchronous value update which is outlined below:

```
void myMibObject::get_request(Request* req, int ind)
{
        //--AgentGen BEGIN=myMibObject::get_request

        // Call my function here, return s, and put it into value by calling set_state(s)
        // Then continue the get request which calls the SnmpDisplayString::get_request(req, ind) below...

        //--AgentGen END
        SnmpDisplayString::get_request(req, ind);
}
```

Some people might want to asynchronously retrieve requested values. An intuitial approach could read as follows:

```
void myMibObject::get_request(Request* req, int ind)
{
        //--AgentGen BEGIN=myMibObject::get_request

        // Callback ptr = &SnmpDisplayString::get_request(req, ind);
        // Put req, pdu and callback_ptr into a message structure and send that in a message to the queue of
another task.
        // do not return any value now (won't be up-to-date)
        if (FALSE)
        //--AgentGen END
        SnmpDisplayString::get_request(req, ind);
}
```

The problem with the above approach is: It will not work with AGENT++ (at least not without additional coding)!

Solution

OK, so how can asynchronous or pseudo-synchronous processing a SNMP request be implemented with AGENT++?
The answer is, there are at least two approaches:

1. Asynchronous (concurrent) processing of all sub-requests with reprocessing of the request when a sub-task associated with sub-request has completed.
2. Pseudo-synchronous processing of each sub-request sequentially.

Asynchronous

The real asynchronous approach has been implemented by AgentX++ in order to be able to concurrently process sub-requests on several sub-agents (or a single multi-threaded sub-agent).

Pseudo-Synchronous

The pseudo-synchronous approach allows queued instrumentation while answering the SNMP request is done synchronously. That means, all sub-requests are processed in the order as they occur in the original SNMP request.

To implement this solution you need:

1. An **OidList<RequestID>** to associate the processed Request...
2. The Request pointer which extends Synchronized and can therefore by used to wait with timeout until another thread notifies it.
3. A pointer to the RequestList to be able to lookup the pending Request to be notified when the instrumentation code finishes.

```
int agentppTestRowCreation::commit_set_request(Request* req, int ind)
{
        //--AgentGen BEGIN=agentppTestRowCreation::commit_set_request
        unsigned long val;
        Vbx vb(req->get_value(ind));
        vb.get_value(val);
        Oidx rowIndex;
        rowIndex += val;
        if (agentppTestSharedEntry::instance->
            pending_row_ops.find(&rowIndex)) {
          return SNMP_ERROR_COMITFAIL;
        }

        // double check for duplicate row (has already been checked in prepare
        agentppTestSharedEntry::instance->start_synch();
        if (agentppTestSharedEntry::instance->find_index(rowIndex)) {
          agentppTestSharedEntry::instance->end_synch();
          return SNMP_ERROR_COMITFAIL;
        }
        agentppTestSharedEntry::instance->end_synch();

        // allocate index and then register and add the row in a background thread
        if (!agentppTestSharedEntry::instance->allocate_index(rowIndex)) {
          *((Gauge32*)value) = 0;
          return SNMP_ERROR_COMITFAIL;
        }
        else {
          agentppTestSharedEntry::instance->
            pending_row_ops.add(new RequestID(req->get_request_id(), rowIndex));
          // wait until row has been registered at master agent
          if (req->wait(AGENTX_DEFAULT_TIMEOUT*1000)) {
            agentppTestSharedEntry::instance->start_synch();
            agentppTestSharedEntry::instance->remove_row(rowIndex);
            agentppTestSharedEntry::instance->end_synch();
            return SNMP_ERROR_COMITFAIL;
          }
        }
        //--AgentGen END
        return MibLeaf::commit_set_request(req, ind);
}
```

The background thread creates the new row in the **agentppTestSharedEntry** table and then calls the **row_added** method listed below:

```
void agentppTestSharedEntry::row_added(MibTableRow* row, const Oidx& index, MibTable* src)
{
        // The row 'row' with 'index' has been added to the table.
        //--AgentGen BEGIN=agentppTestSharedEntry::row_added
        Oidx rowIndex(index);
        RequestID* req_id = pending_row_ops.find(&rowIndex);
        if (req_id) {
          RequestList* req_list =
            backReference->get_request_list();
          if (req_list) {
            Request* req = req_list->get_request(req_id->get_request_id());
            if (req) {
              // signal success
              LOG_BEGIN(EVENT_LOG | 3);
              LOG("agentppTestSharedEntry: row_added: AgentX row successfully registered (index)(request_id)");
              LOG(rowIndex.get_printable());
              LOG(req_id->get_request_id());
              LOG_END;
              req->notify();
            }
          }
          pending_row_ops.remove(req_id);
        }
        //--AgentGen END
}
```