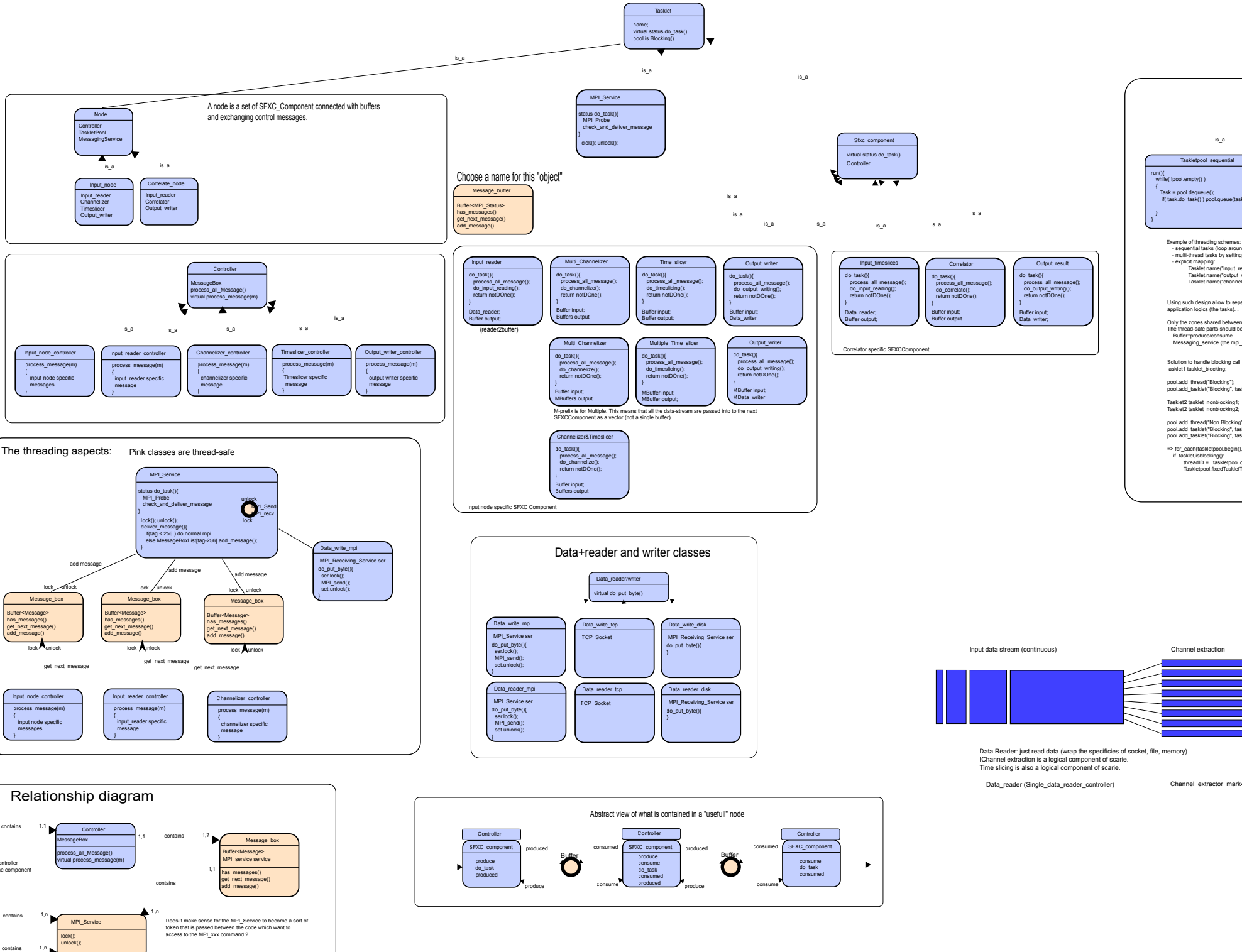
Tasklet
name;
virtual status do_task()
bool is Blocking()
is_a
Node
Controller
TaskletPool
MessagingService
A node is a set of SFXC_Component connected with buffers
and exchanging control messages.
Input_node
Input_reader
Channelizer
Timeslicer
Output_writer
Correlate_node
Input_reader
Correlator
Output_writer
MPI_Service
status do_task(){
MPI_Probe
check_and_deliver_message
}
clok(); unlock();
Sfxc_component
virtual status do_task()
Controller
Choose a name for this "object"
Message_buffer
Buffer<MPI_Status>
has_messages()
get_next_message()
add_message()
Taskletpool_sequential
run(){
while(!pool.empty())
{
Task = pool.dequeue();
if(task.do_task()) pool.queue(task
}
}
Exemple of threading schemes:
- sequential tasks (loop aroun
- multi-thread tasks by setting
- explicit mapping:
Tasklet.name("input_re
Tasklet.name("output_
Tasklet.name("channel
Using such design allow to sepa
application logics (the tasks). .
Only the zones shared between
The thread-safe parts should be
Buffer::produce/consume
Messaging_service (the mpi_
Solution to handle blocking call
asklet1 tasklet_blocking;
pool.add_thread("Blocking");
pool.add_tasklet("Blocking", tas
Tasklet2 tasklet_nonblocking1;
Tasklet2 tasklet_nonblocking2;
pool.add_thread("Non Blocking"
pool.add_tasklet("Blocking", tas
pool.add_tasklet("Blocking", tas
=> for_each(taskletpool.begin(),
if tasklet.isblocking():
threadID = taskletpool.c
TaskletPool.fixedTasklet
Controller
MessageBox
process_all_Message()
virtual process_message(m)
Input_node_controller
process_message(m)
{
input node specific
messages
}
Input_reader_controller
process_message(m)
{
input_reader specific
message
}
Channelizer_controller
process_message(m)
{
channelizer specific
message
}
Timeslicer_controller
process_message(m)
{
Timeslicer specific
message
}
Output_writer_controller
process_message(m)
{
output writer specific
message
}
Input_reader
do_task(){
process_all_message();
do_input_reading();
return notDOne();
}
Data_reader;
Buffer output;
(reader2buffer)
Multi_Channelizer
do_task(){
process_all_message();
do_channelize();
return notDOne();
}
Buffer input;
Buffers output
Time_slicer
do_task(){
process_all_message();
do_timeslicing();
return notDOne();
}
Buffer input;
Buffer output;
Output_writer
do_task(){
process_all_message();
do_output_writing();
return notDOne();
}
Buffer input;
Data_writer
Multi_Channelizer
do_task(){
process_all_message();
do_channelize();
return notDOne();
}
Buffer input;
MBuffers output
Multiple_Time_slicer
do_task(){
process_all_message();
do_timeslicing();
return notDOne();
}
MBuffer input;
MBuffer output;
Output_writer
do_task(){
process_all_message();
do_output_writing();
return notDOne();
}
MBuffer input;
MData_writer
M-prefix is for Multiple. This means that all the data-stream are passed into to the next
SFXCComponent as a vector (not a single buffer).
Channelizer&Timeslicer
do_task(){
process_all_message();
do_channelize();
return notDOne();
}
Buffer input;
Buffers output
Input node specific SFXC Component
Input_timeslices
do_task(){
process_all_message();
do_input_reading();
return notDOne();
}
Data_reader;
Buffer output;
Correlator
do_task(){
process_all_message();
do_correlate();
return notDOne();
}
Buffer input;
Buffer output
Output_result
do_task(){
process_all_message();
do_output_writing();
return notDOne();
}
Buffer input;
Data_writer;
Correlator specific SFXCComponent
The threading aspects:
Pink classes are thread-safe
MPI_Service
status do_task(){
MPI_Probe
check_and_deliver_message
}
lock(); unlock();
deliver_message(){
if(tag < 256) do normal mpi
else MessageBoxList[tag-256].add_message();
}
unlock
MPI_Send
MPI_recv
lock
Data_write_mpi
MPI_Receiving_Service ser
do_put_byte(){
ser.lock();
MPI_send();
set.unlock();
}
add message
add message
add message
lock
unlock
Message_box
Buffer<Message>
has_messages()
get_next_message()
add_message()
get_next_message
get_next_message
get_next_message
Input_node_controller
process_message(m)
{
input node specific
messages
}
Input_reader_controller
process_message(m)
{
input_reader specific
message
}
Channelizer_controller
process_message(m)
{
channelizer specific
message
}
Data+reader and writer classes
Data_reader/writer
virtual do_put_byte()
Data_write_mpi
MPI_Service ser
do_put_byte(){
ser.lock();
MPI_send();
set.unlock();
}
Data_write_tcp
TCP_Socket
Data_write_disk
MPI_Receiving_Service ser
do_put_byte(){
}
Data_reader_mpi
MPI_Service ser
do_put_byte(){
ser.lock();
MPI_send();
set.unlock();
}
Data_reader_tcp
TCP_Socket
Data_reader_disk
MPI_Receiving_Service ser
do_put_byte(){
}
Input data stream (continuous)
Channel extraction
Data Reader: just read data (wrap the specificies of socket, file, memory)
IChannel extraction is a logical component of scarie.
Time slicing is also a logical component of scarie.
Data_reader (Single_data_reader_controller)
Channel_extractor_mark
Relationship diagram
Sfxc_component
Controller
virtual status do_task()
1,1
contains
1,1
Controller
MessageBox
process_all_Message()
virtual process_message(m)
1,1
contains
1,?
Message_box
Buffer<Message>
MPI_service service
has_messages()
get_next_message()
add_message()
1,1
contains
A sfxc_compo contains one and only one controller
A controller is contained by one and only one component
Data_write_mpi
MPI_Service service
1,1
contains
1,n
MPI_Service
lock();
unlock();
1,n
Does it make sense for the MPI_Service to become a sort of
token that is passed between the code which want to
access to the MPI_xxx command ?
Data_reader_mpi
1,1
contains
1,n
Abstract view of what is contained in a "usefull" node
Controller
SFXC_component
produce
do_task
produced
produced
Buffer
produce
consumed
Controller
SFXC_component
produce
consume
do_task
consumed
produced
produced
Buffer
consumed
Controller
SFXC_component
consume
do_task
consumed
consume