



IPS Framework Version 2.0 Changes, Improvements, and Ongoing Work

**Wael Elwasif,
Samantha Foley
And
The IPS Framework Team**

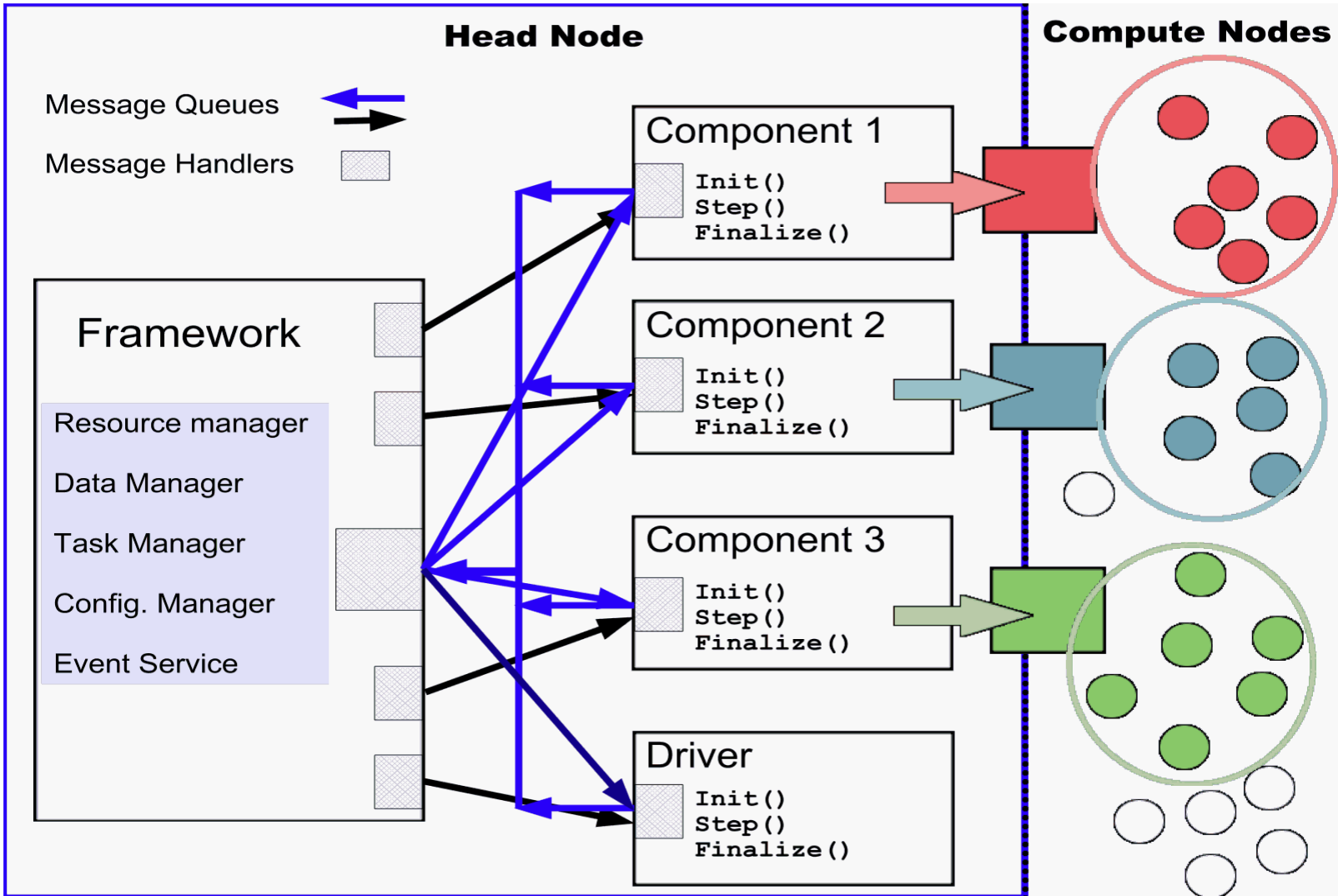
Outline

- The concurrent IPS execution model.
- Changes to the IPS Services API.
- New IPS capabilities.
- Changes to the IPS configuration.
- Ongoing work

Concurrent IPS Execution Model

- Framework and Component wrappers execute concurrently in *separate* processes.
- Framework supports executing multiple concurrent simulations simultaneously (distinct **SIM_NAME** and **SIM_ROOT** config. variables).
- Components and framework communicate using FIFO queues.
- Framework & Components processes execute on the head nodes.
 - Underlying Applications launched on the compute nodes.

IPS Execution Environment



New IPS Capabilities: Concurrent Execution

- Non-blocking component method invocation:
 - Mainly used by the driver to activate two or more physics component methods simultaneously.
 - Shared access to plasma state file(s).

```
rf_comp = self.services.get_port('RF')
nb_comp = self.services.get_port('NUBEAM')
call_id1 = self.services.call_nonblocking(rf_comp, 'step')
call_id2 = self.services.call_nonblocking(nb_comp, 'step')
call_list = [call_id1, call_id2]
call_retval = self.services.wait_call_list(call_list)
print 'rf_comp.step() returned :', call_retval[call_id1]
print 'nb_comp.step() returned :', call_retval[call_id2]
```

Concurrent Execution (2)

- Non blocking execution of physics applications
 - Each instance may run in a separate directory.
 - `launch_task()` is now **ALWAYS** non blocking.

```
cwd = self.services.get_working_dir()
task_id = self.services.launch_task(self.NPROC,
                                     cwd, aorsa_bin,
                                     logfile='log.aorsa')
retcode = self.services.wait_task(task_id)
# Or wait_task_list([task_id])
```

IPS Services API Changes

Sequential IPS	Concurrent IPS
<pre>getPort(port_name)</pre>	<pre>get_port(port_name)</pre>
<pre>call(target_component, method_name, *arg)</pre>	<pre>call_retval = call(target_component, method_name, *args) call_id = call_nonblocking(target_omponent, method_name, *args) call_retval = wait_call(call_id, block) call_retval_dict = wait_call_list(call_id_list, block)</pre>
<pre>launchTask(executable, *args, **keywords)</pre>	<pre>task_id = launch_task(nproc, working_dir, executable, *args, **keywords) ret_val = wait_task(task_id) ret_val = wait_task_nonblocking(task_id) ret_val_dict = wait_tasklist(task_id_list) kill_task(task_id) kill_all_tasks()</pre>
<pre>GetGlobalConfigParameter(param)</pre>	<pre>get_config_param(param) set_config_param(param, value)</pre>

IPS Services API changes (2)

Sequential IPS	Concurrent IPS
<code>getTimeLoop ()</code>	<code>get_time_loop ()</code>
<code>getWorkingDirectory (component)</code>	<code>get_working_dir ()</code>
<code>stageInputFiles (component, inputFileList)</code>	<code>stage_input_files (input_file_list)</code>
<code>stageOutputFiles (component, timeStamp, outputFileList)</code>	<code>stage_output_files (timeStamp, file_list)</code>
<code>stageCurrentPlasmaState (component)</code>	<code>stage_plasma_state ()</code>
<code>updatePlasmaState (component)</code>	<code>update_plasma_state ()</code> <code><i>merge_current_plasma_state (</i></code> <code><i> partial_state_file)</i></code>

Note: `self` is typically used for the `component` argument in sequential services invocation.

New IPS Capabilities: Event Service

- *Event topic* based, asynchronous publish/subscribe capability for data exchange.
- An *event* is published to a topic is received by all subscribers to that topic, when they poll for events.
- API:
 - `publish(topicName, eventName, eventBody)`
 - `subscribe(topicName, callback)`
 - `process_events(self)`
 - `unsubscribe(topicName)`
- `eventBody`: An “*agreed upon*” python dictionary.
- `callback method : name(topicName, theEvent)`

New IPS Capabilities: logging

- Multi-level logging capability.
 - New services calls (simplest case: one string argument):

<code>debug()</code>	<code>info()</code>	<code>warning()</code>
<code>error()</code>	<code>critical()</code>	<code>exception()</code>
- Simulation-level configuration parameter, `LOG_LEVEL`, controls the output (default value: `WARNING`).
- Required config parameter: `LOG_FILE`
- Component level parameter, `LOG_LEVEL`, overrides the simulation-level value for that component.
- `exception()` is called from within a python `except:` clause to print detailed trace information.



New IPS Capabilities: Multiple Concurrent Executions

- Framework can manage multiple concurrent simulations (multiple `--config` command line options).
- Allows for efficient resource utilization by executing similar simulations “*out of phase*”.
- No change is needed in the individual simulation configuration.
- Events are confined to individual simulations.
- Requirement: distinct `SIM_ROOT` and `SIM_NAME` configuration parameters.

New IPS Capabilities Checkpoint - Restart

- Added component methods:
 - `checkpoint()`
 - `restart()`
- Added service methods:
 - `save-restart_files()`
 - `get-restart_files()`
- Intended for coordinated checkpoint (initiated by the driver)

Changes to IPS Configuration

- Platform configuration file:
 - Capture system-wide parameters (**MPIRUN**, **PHYS_BIN_ROOT**, **PORTAL_URL**, **RUNID_URL**, ..etc)
 - Can be overridden in individual simulations config files.
- Framework log file:
 - Populated when option `--debug` is used on the command line, otherwise defaults to warnings.
- Component-level **PLASMA_STATE_FILES** parameter:
 - Subsets the global entry for optimal data movement and archival.
- Allow wildcard (*) specification of input/output files.

Miscellaneous Coding Issues

- **DO NOT** call `sys.exit()` in your component, either return to caller, or **raise** a python exception.
- Failure in calls to `services.XYZ` is communicated using Python exceptions. Example:

```
try:
    self.services.XYZ(...)
except Exception:
    self.services.exception("Error occurred in XYZ()")
    raise
```

Ongoing Work

- Plasma state sharing across concurrent components:
 - `merge_current_plasma_state()`
 - Is it enough? What else is needed? What about other files?
 - Need more physics use cases.
- Finishing checkpoint/restart machinery.
- Talk to us if more framework capabilities are needed for the physics runs
 - Communicating “*physics events*”.
 - Adaptive time stepping.
 - ...

More Information

- See presentation in Coding Camp section.
- Samantha S. Foley, Wael R. Elwasif, Aniruddha G. Shet, David E. Bernholdt, and Randall Bramley, “***Incorporating Concurrent Component Execution in Loosely Coupled Integrated Fusion Plasma Simulation***”, in Component-Based High-Performance Computing (CBHPC) 2008, 2008, (extended abstract).
- Wael Elwasif, David E. Bernholdt, Aniruddha G. Shet, Samantha S. Foley, Randall Bramley, Donald B. Batchelor, and Lee A. Berry, “***The Design and Implementation of the SWIM Integrated Plasma Simulator***”, in The 18th Euromirco International Conference on Parallel, Distributed and Network-Based Computing (PDP 2010), 2010, submitted.