iSAM
Software Architecture

DRAFT

Author:
Scott Rasmussen
Zaita Design
scott.rasmussen@zaita.com

Table of Contents

Document History

Version	Description	Author	Date
1.0	*Initial version - Draft*	S.Rasmussen	13/06/2012
1.1	*Modification of diagrams, explanation of states*	S.Rasmussen	12/07/2012
1.2	*Updating diagram to show modifications to states*	S.Rasmussen	28/02/2013
	Updating Development environment		

iSAM Overview

iSAM will be the predecessor to the original CASAL modelling application that was developed approx 10 years ago. It'll be developed using modern technology and sound development techniques to ensure the code-base is maintainable and integrity of the application can be verified at any time.

iSAM II's architecture will be based on the design mentality behind the Spatial Population Model (SPM - http://www.niwa.co.nz/fisheries/tools-resources/spm-spatial-population-model). This model is highly modular with the code developed in small light-weight classes that are easily recognisable and follow a well designed pattern of construction.

SPM's code base is nearly 5 years old, and is as-maintainable now as it was when it was first developed. The techniques used to develop SPM are proven, and will be applied to the development of iSAM.

Supported Software Requirements

iSAM will be developed as a native 64 bit (x64) application with no 32 bit (x86) support. Compilation and testing will not be done on a x86 platform.

The processor families supported will be Intel and AMD processors that conform to the AMD64 (x64) specification. PowerPC and ARM processors will not be supported.

Operating Systems supported will be Windows 7 x64 and Linux x64. All other Operating System variants (BSD, Unix, OSX, Android, IOS) will not be supported.

Development Environment

iSAM will primarily be developed on:

Operating Systems

1. Microsoft Windows 7 - x64
2. OpenSuSe 12.2 (Mantis) - x64

Development Environment

Note: It is expected that during the development of iSAM some of these tools/libraries will be upgraded to more current versions as they are released.

Windows

- TDM-GCC 4.7.X (http://tdm-gcc.tdragon.net/)
- Notepad++ (http://notepad-plus-plus.org/)
- AQTime3 – Profiling
- AMD Code Analyst
- AMD App Profiler

Linux

- GCC/G++ 4.7.X
- Valgrind

Both

- Eclipse Juno (http://www.eclipse.org/)
- Boost 1.53.X (http://www.boost.org/)
- CMake 2.8.X (http://www.cmake.org/)
- GNU Debugger (GDB)

Coding Style

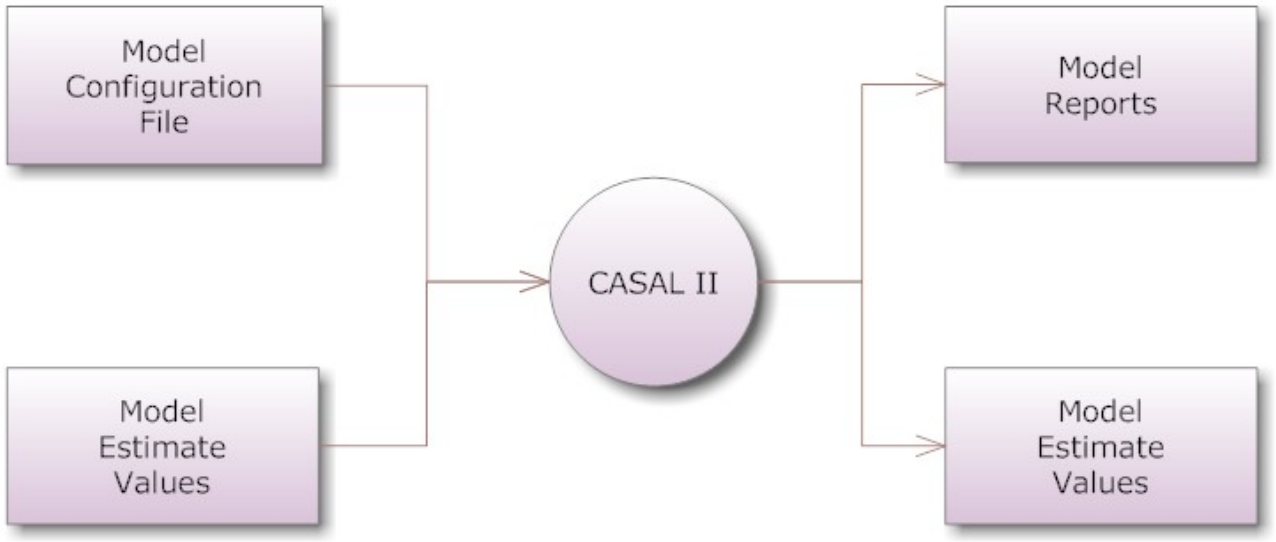
While it's going to be a step away from the style used for SPM, iSAM will use the Google Coding style (http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml). Google provides a handy script to parse source code and highlight errors that do not match their coding style.

Spelling of variable names, classes etc will all be done using British English.

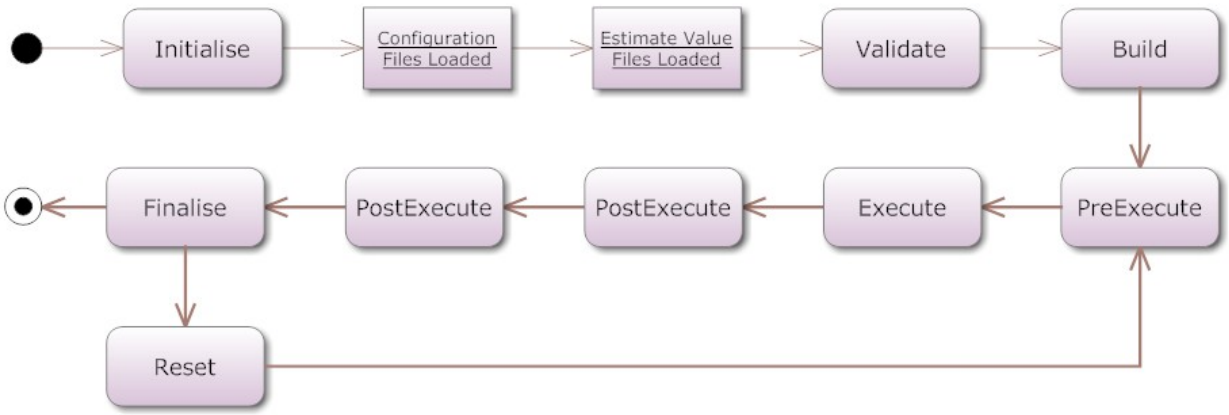
*Note: The only deviation from this style is the use of *.cpp instead of *.cc for filename extensions.*

High-Level Design

Level 0 Data-Flow-Diagram



State-Transition Diagram



State Descriptions

Initialise

The initialise state is the creation of components and parsing of configuration files.

Tasks completed:

- Parse command line
- Parse configuration file
- Load plugins
- Load estimate values from input files

Validate

The validate state is used for checking user supplied options and parameters to ensure they fit within hard-limits imposed by the software.

Build

The build phase is where the system will build relationships between objects that rely on each other. Because validation has been completed, each object in it's self-contained configuration is ok.

This phase generally assigns values to shared_ptrs<> for objects so they don't need to do lookups of objects during execution phases.

PreExecute

Pre-Execution happens in a timestep. Before the time-step objects that require a pre-execution call will then be called. This is usually done by Derived values or reports

Execution

The model is executing the processes

PostExecute

Finalise

Finalise will happen after all iterations of the model have been completed. In an estimation run this means some kind of convergence with the minimiser.

Software Components

Utilities Library

A stand-alone library for doing common functions within NIWA's modelling applications will be developed for iSAM II.

This library will offer functionality like: Logging, Error handling, double comparison, vector/map manipulation, string manipulation.

Configuration File Parser

A stand-alone library for loading and parsing configuration files for NIWA's modelling applications will be developed for iSAM II. The code from this will be mostly inherited from SPM and Random Station, but will be largely re-factored to fit into a stand-alone library.

Equation Parser

iSAM will use an open source equation parser with modifications to make it compatible with ADMB. It's likely that two equation parsers will exist within iSAM and a runtime decision will be made about which one to picked based on the runtime parameters.

Minimisers

iSAM will support 3 minimisers natively,

- DE Solver
- GammaDiff
- ADMB

While ADMB does need tight integration with iSAM II, the other 2 minimisers do not. These minimisers will be extracted into seperate shared libraries and built as stand-alone components.

State-Model

As the state model for iSAM is quite transferable across different modelling applications it will be also extracted into it's own stand-alone library.

Plugin Architecture

Dynamic-Library

Developers with enough competence in C++ will be able to develop and load their own plugins by building shared-libraries and specifying the location of these within their configuration files.

An expected inclusion section of someone's plugin would be:

```
#include <niwa/iSAM/process.h>
#include <niwa/iSAM/selecvity.h>

class myNewProcess : public niwa::iSAM::process {
public:
 void validate() { }
 void build() { }
 void execute() { }
}
```

Difficulty for user to develop: High

Execution speed: Fast

Command-Line Executable

Some components of the application will be replacable with command line applications that take specific arguments and return a single result (e.g Selectivities/Layers).

A specification will be developed to allow people to build and specify stand-alone executable based plugins for specific functionality within iSAM II.

The upside to this approach is that the user can specify any type of executable they wish, developed in any language, including shell-scripts. The application will simply do an exec() call on that object and intepret the result.

Difficulty for user to develop: Low

Execution speed: Slow

Equation Parser

iSAM will have an inbuilt equation parser for handling equations specified natively in the configuration file.

A valid example equation would be: 3^x * 2

Where the user is able to bind 'x' to an internal parameter inside iSAM II.

Difficulty for user to develop: Low

Execution speed: Slow

OpenCL Kernel

Note: This method was investigated, but at this point in time will not persued. Keeping this information in here as a possible future expansion of the software.

An OpenCL kernel is a small file containing a vectorised C++ snippet of code. When iSAM starts it's able to load the kernel and compile it against either a GPU (when one is present) or a CPU.

The speed benefits from using OpenCL on a GPU can be enormous because of it's natively ability to work with multi-dimensional sets of data. Other benefits are the ability to send someone a code-snippet and have iSAM compile it on the fly for execution, so platform indepdence is not an issue.

The major downsides to this approach are development architecture and difficulty. Developing an application around an OpenCL architecture either has to be very modular or completely ingrained in the application's structure. Difficulty in developing for OpenCL is very high, especially among people who are not familiar with C++ syntax or vector languages (e.g R/S+).

Difficulty for user to develop: High

Execution speed: Extremely Fast