iSAM
Software Architecture

DRAFT

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

Table of Contents

Document History....3
iSAM Overview....3
Supported Software Requirements....3
Development Environment....4
  Operating Systems....4
  Development Environment....4
    Windows....4
    Linux....4
    Both....4
  Coding Style....4
High-Level Design....5
  Level 0 Data-Flow-Diagram....5
  State-Transition Diagram....5
State Descriptions....6
  StartUp....6
  Validate....6
  Build....6
  Verify....6
  PreExecute....7
  Execute....7
  PostExecute....7
  IterationComplete....7
  Reset....7
  Finalise....7
Software Components....8
  Utilities Library....8
  Configuration File Parser....8
  Equation Parser....8
  Minimisers....8
  State-Model....9
Plugin Architecture....9
  Dynamic-Library....9
  Command-Line Executable....9
  Equation Parser....10
  OpenCL Kernel....10
Population Processes....11
  Category Shifting....11
  Recruitment....11
  Mortality....11
  Ageing/Growth....11
Software Integrity....12

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		
1.3	*Update to show functionality created as part of phase 1*	S.Rasmussen	05/07/2013

iSAM Overview

iSAM is the successor to the CASAL modelling application that was developed approximately 10 years ago. It has been developed using modern technology and current best practice development techniques to ensure maintainability and integrity.

ISAM's architect is based on the the design mentality behind the Spatial Population Model (SPM - http://www.niwa.co.nz/fisheries/tools-resources/spm-spatial-population-model). The code base is highly modular with code developed in small light-weight objects that are easily recognisable.

The SPM code base was developed in 2007 and is still maintainable today as the code follows a well documented coding standard and a simplistic layout for objects. 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 x64 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

- Microsoft Windows 7 (x64)
- 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 Kepler (http://www.eclipse.org/)
- Boost 1.54.X (http://www.boost.org/)
- CMake 2.8.X (http://www.cmake.org/)
- GNU Debugger (GDB)
- Python 2.X

Coding Style

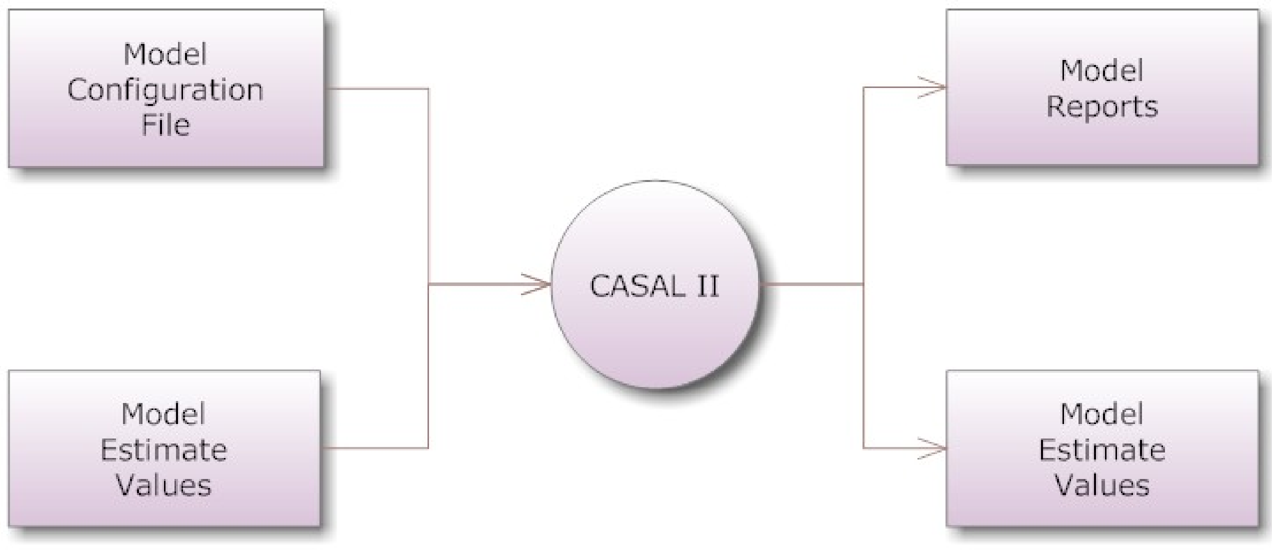
While it's going to be a step away from the style used for SPM, iSAM will use the Google Coding style (http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml). Google provides a handy script to parse source code and highlight errors that do not match their coding style.

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

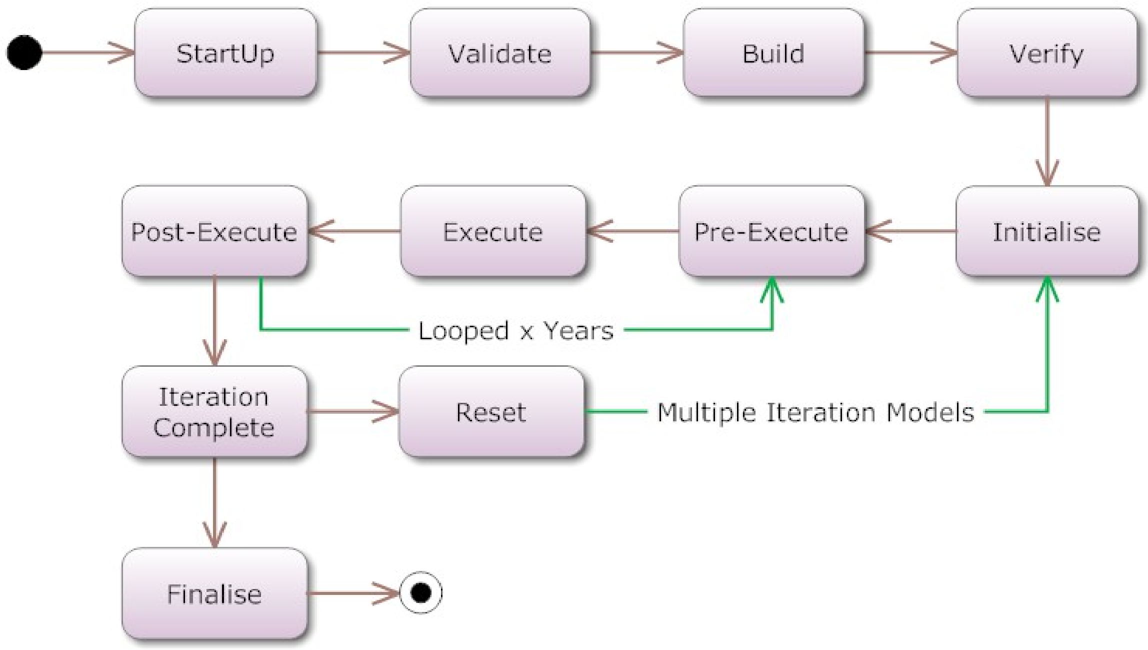
*Note: The only deviation from this style is the use of *.cpp instead of *.cc for filename extensions.*

High-Level Design

Level 0 Data-Flow-Diagram



State-Transition Diagram



State Descriptions

StartUp

The model is in the blank start and the configuration system is loading the configuration files and parsing any extra inputs.

Tasks completed:

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

Validate

All user configurations have been loaded at this point. Now the model will go through every object that has been created and check that the parameters given to them.

This step will ensure every object in the model has sufficient parameters to be executed without causing a system fault or error in the model.

This state will not check the values to ensure they are logical in relation to an actual model. They will only test that they exist and meet minimum requirements to execute a model.

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.

Verify

At this point pre-defined configurations are checked against the model's current configuration to verify if the model makes sense logically. These are business rules being applied to the model to help ensure the output is not garbage.

Note: This state is not executed by default and must be defined as part of the model execution.

Note: This has not been implemented.

PreExecute

Pre-Execution happens at the beginning of a time step. This allows objects to calculate values based on the partition state before any of the other processes in the time step are executed.

Execute

This is the general work method of the model and where all of the processes will be run against the partition.

PostExecute

This is executed at the end of a time step after all of the processes and associated objects have been executed. This is typically used for things like reports and derived quantities.

IterationComplete

This is executed at the end of every model run. This is only useful when the model is in a multiple-iteration mode (e.g MCMC or Estimation). After every model iteration this state is triggered.

Reset

If the model has to run multiple iterations then the reset state is used to reset everything back in to a state where the model can be re-executed without any legacy data remaining.

This state allows us to run multiple iterations of the model without having to re-process the configuration information or de-allocate/re-allocate large amounts of memory.

Finalise

Finalise will happen after all iterations of the model have been completed.

Software Components

Utilities Library

Inside iSAM is a collection of re-usable methods for reading the command line, converting between types, using auto-differentiation types, logging, error handling, double comparison etc.

While not a stand-alone library these methods can easily be extracted for use within other NIWA projects.

Configuration File Parser

The configuration parser was developed from the ground up as a new component for iSAM using ideas inspired by the SPM implementation. While it's not a stand-alone component it is still in a state that allows it to be easily ported to other applications.

Some of the in-built functionality of the configuration file parser is a "parameters" architecture that allows for quick retrieval and validation of user supplied parameters with type-conversions and validations. The configuration parser also has the ability to track what file and line a particular parameter was defined to be used for error reporting.

Note: it is expected that SPM will move to the same coding standard as iSAM in the future and one of the first components that will make a migration back from iSAM will be the configuration parsing system.

Equation Parser

It is hoped that iSAM will have the ability to use inline equations at specified locations within the configuration file. This will give the user the ability to create new objects without having to do code modifications.

Note: This has not been implemented.

Minimisers

iSAM will support 3 minimisers natively,

- DE Solver
- GammaDiff
- BetaDiff

The minimisers cannot all be run from the same binary. As BetaDiff requires the entire model to be compiled with special parameters that override all internal "double" types the overhead introduced by this makes it not suitable for GammaDiff or DE Solver.

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

Note: No plugin architect has been implemented.

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

Population Processes

ISAM supports a number of population processes. While there is a large number of individual processes the general purpose of these can be broken down in to a few different types.

Category shifting, recruitment, mortality and ageing/growth.

Category Shifting

These processes are responsible for moving part of the population from 1 category in to another.

Implemented in iSAM there is:

- Rate

Recruitment

These processes are responsible for the introduction of new population members.

Implemented in iSAM there is:

- Constant rate
- Beverton-Holt

Mortality

These processes are responsible for the removal of population members.

Implemented in iSAM there is:

- Constant rate
- Event

Ageing/Growth

These processes are responsible for moving population members up through ages/lengths. They work similar to category shifting except only work within a single category.

Implemented in iSAM there is:

- Ageing

Software Integrity

One of the key focusses in the iSAM development is the emphasis on software integrity. It's hugely important to ensure results coming from user models are consistent and correct.

As part of this we utilise unit tests to check individual components of the software and run entire models verifying results.

ISAM uses:

- Google testing framework
- Google mocking framework

As at 5th July 2013 the unit test output was:

```
[==========] Running 19 tests from 4 test cases.
[----------] Global test environment set-up.
[----------] 7 tests from BasicModel
[ RUN      ] BasicModel.Observation_Abundance
[       OK ] BasicModel.Observation_Abundance (3 ms)
[ RUN      ] BasicModel.Accessors_Cached_CombinedCategories
[       OK ] BasicModel.Accessors_Cached_CombinedCategories (1 ms)
[ RUN      ] BasicModel.Processes_Constant_Recruitment
[       OK ] BasicModel.Processes_Constant_Recruitment (1 ms)
[ RUN      ] BasicModel.Processes_Mortality_Event_No_Penalty
[       OK ] BasicModel.Processes_Mortality_Event_No_Penalty (1 ms)
[ RUN      ] BasicModel.Processes_Mortality_Constant_Rate
[       OK ] BasicModel.Processes_Mortality_Constant_Rate (1 ms)
[ RUN      ] BasicModel.Processes_Maturation_Rate_Constant_One_Selectivity
[       OK ] BasicModel.Processes_Maturation_Rate_Constant_One_Selectivity (1 ms)
[ RUN      ] BasicModel.Processes_Ageing
[       OK ] BasicModel.Processes_Ageing (1 ms)
[----------] 7 tests from BasicModel (10 ms total)

[----------] 1 test from PartitionAccessors
[ RUN      ] PartitionAccessors.Category
[       OK ] PartitionAccessors.Category (0 ms)
[----------] 1 test from PartitionAccessors (0 ms total)

[----------] 10 tests from Selectivities
[ RUN      ] Selectivities.LogisticProducing
[       OK ] Selectivities.LogisticProducing (0 ms)
[ RUN      ] Selectivities.Logistic
[       OK ] Selectivities.Logistic (0 ms)
[ RUN      ] Selectivities.KnifeEdge
[       OK ] Selectivities.KnifeEdge (0 ms)
[ RUN      ] Selectivities.InverseLogistic
[       OK ] Selectivities.InverseLogistic (0 ms)
[ RUN      ] Selectivities.Increasing
[       OK ] Selectivities.Increasing (0 ms)
[ RUN      ] Selectivities.DoubleNormal
[       OK ] Selectivities.DoubleNormal (0 ms)
[ RUN      ] Selectivities.DoubleExponential
[       OK ] Selectivities.DoubleExponential (1 ms)
[ RUN      ] Selectivities.Constant
[       OK ] Selectivities.Constant (0 ms)
[ RUN      ] Selectivities.AllValuesBounded
[       OK ] Selectivities.AllValuesBounded (0 ms)
[ RUN      ] Selectivities.AllValues
[       OK ] Selectivities.AllValues (0 ms)
[----------] 10 tests from Selectivities (2 ms total)

[----------] Global test environment tear-down
[==========] 18 tests from 4 test cases ran. (14 ms total)
[  PASSED  ] 18 tests.
```

A basic coverage of the currently implemented processes and selectivities has already been achieved.