Casal2
Software Architecture

v2019.12

Author:
Scott Rasmussen
scott@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		
1.3	*Update to show functionality created as part of phase 1*	S.Rasmussen	05/07/2013
V2016.1	*Update to reflect released version 1.0 of Casal2*	S.Rasmussen	20/01/2016
V2019.12	*Update to reflect released version 1.0 of Casal2*	S.Rasmussen	03/12/2019

Casal2 Overview

Casal2 is NIWA’s new integrated assessment tool for modelling population dynamics of marine species, including fishery stock assessments. Casal2 expands functionality and maintainability over its predecessor, CASAL. Casal2 can be used for quantitative assessments of marine populations, including fish, invertebrates, marine mammals, and seabirds.

Casal2 has been developed using modern C++ standards and current best practice software development techniques to ensure maintainability and integrity. Casal2's architecture is based on the high cohesion low coupling philosophy. Components are isolated into logical separations both on disk and in code. There is a strong correlation between the model definition language (MDL) input file and the code structure. The code base is highly modular with code developed in small light-weight objects that are easily and extensible.

Solution Considerations

Goals and Guidelines

The goals of Casal2 are to:

- Produce a high-speed modelling framework
- Allow for easy customisation of models using a Model Definition Language for quick model development
- Provide clear error handling and messages to the user
- Use modern tools and techniques to produce high quality code
- Focus on code quality and integrity of results as the highest priorities
-

The guidelines for the Casal2 development are:

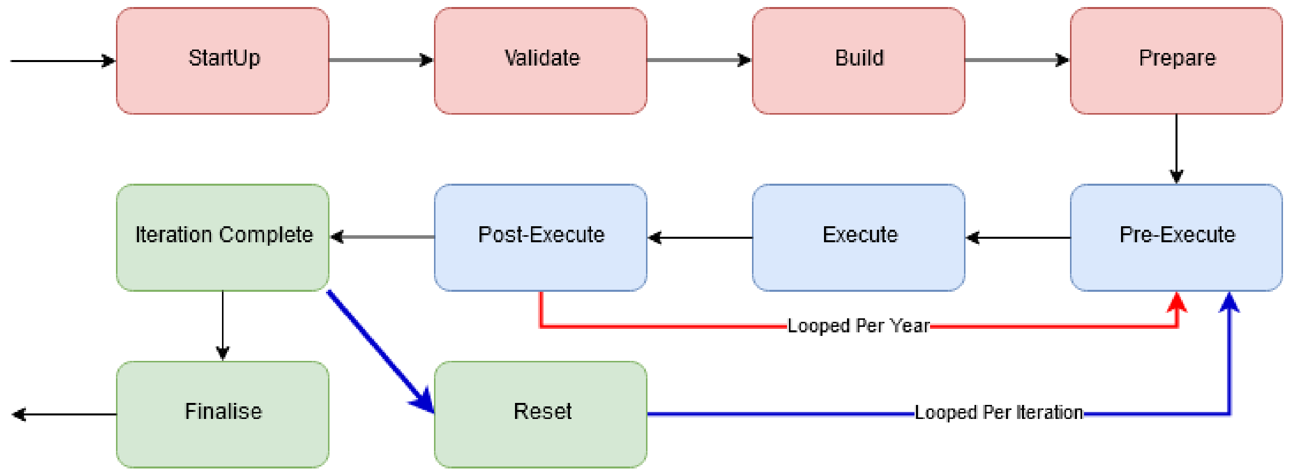
- Use the latest technologies and tools
    - Automated build systems will be used to provide maximum code coverage for new changes by running all tests on every push to the git repository.
    - Unit testing will be used to ensure code integrity is always maintained.
    - New C++ language features and specifications will be adopted quickly to simplify code, increase performance, and reduce bugs.

Casal2's architecture has been designed to promote high-speed, low overhead development based on Agile software development principles. The principles taken from the Agile manifesto that have been applied to Casal2 are:

1. **Individuals and interactions** over processes and tools
    a. People are free to contribute code directly to the Casal2 code base without overhead. No prioritisation, processes, or large discussions groups/quorums have been used. Emphasis is always on producing new and exciting features.
    b. People are responsible for their code and its quality. Unit tests, Model runs, and the automated build system are used to ensure everyone contributing does so with code that always compiles on the master branch.
2. **Working software** over comprehensive documentation
    a. The code should be written in a way that it is "self-documenting". Code comments should be liberally used to explain what the code is doing.
    b. Unit tests should be used as much as practical to ensure the code functions as expected. Unit tests should also serve as documentation for the code base by demonstrating how the functionality works.
    c. Automated builds and tests are run after every commit to the master branch. Users are required to immediately fix any breaks before they continue.
3. **Customer collaboration** over contract negotiation
    a. Casal2 encourages new ideas and concepts. Adding new objects (processes, observations, etc) can be isolated with no risk to the larger system functionality.
    b. Casal2 encourages the prototyping of new research concepts by using a de-coupled, modular approach to Software Development.
    c. Small features can be added to Casal2 quickly to allow for rapid iteration and feedback.
4. **Responding to change** over following a plan
    a. The Casal2 Architecture has been designed to be flexible, allowing significant changes to be implemented with little risk or increased complexity.
    b. Casal2's development has been iterative, focussing on the next required functionality only at any given point, and following good software architecture principles and design patterns to ensure that the amount of technical debt generated is minimized.
    c. The version control software git has been used during the development of Casal2.

Solution Architecture

State Machine Diagram



State Descriptions

StartUp

The model is in the blank state and the configuration system is loading the configuration files and parsing any special command line parameters. Multiple input files can be loaded during this phase (e.g., observation data, estimated values, MPD, MCMC samples, MCMC objectives).

Validate

The model will go through every object and validate the parameters. This step will ensure every object in the model has enough 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 business rules; it will test only that the objects and parameters exist and meet minimum validation requirements.

At the end of the validate stage each object should be internally consistent. No lookups or external references are allowed during this stage.

Build

The system will build the relationships between objects. Objects are now allowed to communicate with each other. Where caches of objects are needed that rely on external objects, this is when they will be built. This phase assigns values to pointers for objects so they don't need to do lookups of objects during the execution phases.

Note: It is common for objects to call "Reset" during this phase to handle building of objects that will need to be reset after each iteration

Prepare

The system will call the models with a Prepare flag for them to build and/or write any headers that need to be done only once.

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 processes will be run against the partition.

By default, Casal2 uses a standard time-step object where a year is constructed of multiple time-steps run in sequence. Each time step has one or more processes executed in sequence.

PostExecute

This is executed at the end of a time step after all 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 primarily used 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 must perform multiple iterations, then the Reset state is used to reset everything. The model can then be re-executed without any previously calculated results 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 happens after all iterations of the model have been completed. Any reports that need to be finalised and written to disk will be done at this point.

Architectural Strategies

Design

Casal2 has been designed as a state machine. During the execution of a model, the system will transit through different states, executing dependent objects as it goes. The system tries to maintain the concept of a state machine at all points throughout the execution.

All executable objects in Casal2 inherit from a base::Object class that provides the foundations for parameter loading and addressable access. This is a concept taken from Java.

Memory Management Policies

Casal2 prefers the use of Raw Pointers, not shared_ptr or unique_ptr. When the development of Casal2 started, performance profiling of shared_ptrs was done and the overhead was 50-100% for model runs. Because of this, it was decided that Casal2 would be developed using Raw Pointers.

Note: It is recommended that some aspects of Casal2 be migrated to shared_ptrs as the adoption of threading architectures are implemented. Compiler optimisations of shared_ptrs have improved significantly and it is likely that the overhead is no longer applicable.

Objects in Casal2 are "owned" by their respective Managers. It is the responsibility of each Manager to free the memory of its child objects. All Managers are owned by the core "Model" object and it is responsible for destructing the Managers. Through this strict hierarchy of ownership, Casal2 has been successfully able to avoid memory leaks while using raw pointers.

Threading Policies

Casal2 is not thread safe, nor is the model re-entrant. Do not attempt to execute model runs in different threads as this will cause race conditions and undefined behaviour.

Casal2 does use threading for reporting. The reports are printed to stdout/disk in a separate thread to the main execution. During execution, reports write their data to a stringstream cache. The report then uses an atomic flag to signal that it is ready for printing. The printing thread then locks the report object and prints the cache to stdout/disk.

Error Detection and Reporting

Casal2 uses a collection of C++ macros for error reporting to the user. The macros provide access to the __LINE__ and __FILE__ values which are used to print the source code location of the error. The file loading framework in Casal2 stores the file name and line of all input parameters. This information is displayed back to the user as part of the error messaging.

Where errors are not immediately detrimental, Casal2 allows the execution to continue and other errors to be detected and logged. When execution gets to a critical point, the error condition of the model is checked. If errors exist, the top 10 are printed to the console and execution finishes. This functionality has been designed to allow the user to see and fix multiple errors in a single iteration.

Performance Engineering

Casal2 has been designed to be as fast as possible. When a model is parsed from the Model Definition Language, Casal2 will create runtime objects and links only for objects that have been specified. There is no excess allocation of memory for objects or data.

The code base is often profiled to ensure execution states are not allocating memory or having excessive cache-misses. Profiling is not, nor is it likely possible, integrated with the automated build system. Profiling has been done using AMD CodeXL,Valgrind, gProf, Very Sleepy and

GCC/MSVC.

The Casal2 execution states (reset, pre-execute, execute, post-execute) are not allowed to allocate new memory. Where possible, the allocation of memory is limited to the validation and build states only as these states are run-once regardless of the run mode.

Threading is used to print reports to stdout/disk as I/O is a resource-intensive operation. Cached writes are done during the execution of the model, which allocates memory. In the future the developers may explore how to reserve the amount of memory required for reports in the build phase, but this has not been implemented.

Casal2 is tested with different compiler settings but favours the use of Optimisation Level 2 on gcc as this does not interfere with the results of the models. Level 3 optimisation changes the math routines for speed, but this comes at a cost to accuracy and reproducibility.

Development Methods

Casal2 is developed using:

- Agile software development methodologies/principles
- Object-oriented development using C++
- Test-driven development using the Google Mock and Test frameworks

Casal2 uses the Google C++ coding style: https://google.github.io/styleguide/cppguide.html

*Exception: *.cc is replaced with *.cpp for source files*

Casal2 uses a hub-n-spoke approach to construct the final package. The hub is the front-end application that loads a collection of shared libraries (spokes). Each of the spokes is a compilation of the Casal2 code base with different characteristics. The different spokes include the automatic differentiation libraries, the standard Casal2 library, and unit test library.

The development of Casal2 relies heavily on well-defined software design patterns. Specifically, the factory, façade, observer, strategy, and state patterns are used.

Application Architecture

Operating Systems

Casal2 can be built and run on Linux and Windows 64-bit operating systems. Casal2 may build and run on other operating systems (BSD, Unix, MacOS, Android, iOS), but this has not been tested.

Casal2 runs on 64-bit systems, with no support for 32-bit systems. The supported CPUs (processor families) are Intel and AMD processors that conform to the AMD64 (x64) specification. Sparc, PowerPC, and ARM processors are not supported.

Build Wrapper

Casal2 has a custom set of Python3 scripts that act as a build wrapper. Due to the complexity of obtaining, configuring, and compiling the third-party libraries and the Casal2 profiles, a custom wrapper was developed.

The Casal2 build wrapper is in the BuildSystem subdirectory and is accessed through the doBuild.sh (Linux) and doBuild.bat (Windows) files when executed from a terminal/command line.

Installation Packages

Linux

Casal2 is distributed as a .deb (Debian Installer) file and a .tar.gz file. The Casal2 package comes with binaries and shared libraries that need to be installed in the distribution's shared library supported locations.

Windows

Casal2 is distributed as a self-extracting executable installer. The installer is produced as part of the build system using the Inno Setup Compiler.

Development Environments

Casal2 was developed based on the C++11 standard. Because of this, compilers and associated tools must support C++11.

To maintain consistency between Linux and Windows, gcc is the compiler chain for the development of Casal2. This allows the use of near-identical compiler flags and third-party libraries. gcc must be at least version 4.8.5 as this supports the full C++11 specification.

Windows

- MinGW64 gcc/g++ 4.8.5+
- MinGW64 gfortran 4.8.5+

Casal2 comes with components as part of its build system. These are:

- Unix Utils - *nix command line applications for Windows
- cmake 2.8.X

Linux

- gcc/g++ 4.8.5+
- gcc/g++ Fortran 4.8.5+
- cmake 2.8.X+

– Python 3.7+

Note: cmake 3.X+ will work but will provide warnings around the use of IF(TEST) as this has been deprecated because it is now a keyword.

Software Components

Utilities Library

Casal2 includes a collection of reusable methods for reading the command line, converting between types, using automatic differentiation types, logging, error handling, double comparison, etc.

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

Configuration File Parser

The configuration parser was developed from the ground up as a new component for Casal2 using ideas inspired by the SPM implementation. While it is not a stand-alone component it is decoupled enough to allows it to be ported to other applications.

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

Minimisers

Casal2 includes multiple minimisers:

- ADOL-C (automatic differentiation) https://projects.coin-or.org/ADOL-C
- BetaDiff (automatic differentiation)
- CppAD (automatic differentiation) https://coin-or.github.io/CppAD/doc/cppad.htm
- GammaDiff / Numerical Differences
- DESolver – Differential Evolutionary Solver http://www.math-cs.gordon.edu/~senning/desolver/
- Dlib http://dlib.net/

Software Integrity

One of the key objectives for the Casal2 development is the emphasis on software integrity. It is important to ensure that results coming from user models are consistent and correct.

As part of this focus, unit tests are utilized to check individual components of the software and run entire models verifying results.

Casal2 uses:

- Google testing framework
- Google mocking framework
- Models executed as part of the automated build suite using a "modelrunner" build profile

The automated build system for Casal2 runs:

- 15 Windows builds
- 15 Linux builds

Plugin Architecture

Casal2 has a requirement to support plugins of different types (e.g., user scripted processes). No plugin architecture has been developed, but Casal2 does not prohibit the inclusion of libraries supporting the development of processes (etc) in Python/R.

Equation Parser

Casal2 has an inbuilt equation parser for handling equations specified natively in the configuration file.

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

Where x is the name of an addressable in Casal2.

Process Types

Casal2 supports many types of processes. These processes can be summarized into a few types including category shifting, recruitment, mortality, and ageing/growth.

Category Shifting

These processes are responsible for moving part of the population from one category into another.

Recruitment

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

Mortality

These processes are responsible for the removal of population members.

Ageing/Growth

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