Towards Modelling and Verification of Coupler Behaviour in Climate Models

Chinmayi Baramashetru
c.baramashetru@kent.ac.uk
University of Kent
UK

Dominic Orchard
dominic.orchard@cl.cam.ac.uk
University of Cambridge
University of Kent
UK

Abstract

Climate models and earth system models often comprise submodels composed via a 'coupler', a software component that enables interaction between submodel components. The continuous exchange of data through couplers creates the risk of subtle errors propagating across components, potentially distorting scientific conclusions. In this paper, we argue for lightweight formal verification techniques applied at the coupler interface to improve both coupler and model correctness. By enforcing formal contracts on data exchanges, the coupler can act as a checkpoint that detects and prevents certain classes of component-level errors before they propagate between models. We abstract general design principles for couplers and propose verifiable subsystems. Using an example of a real-world bug, we illustrate a hybrid verification strategy that integrates module-level contracts, verified through both static and runtime techniques. We aim to offer a practical pathway for both existing and future couplers, ultimately enabling auditable and formally verifiable couplers.

CCS Concepts: • **Software and its engineering → Software verification and validation**; • **Computing methodologies → Modeling and simulation**; • **Applied computing → Earth and atmospheric sciences**.

Keywords: Climate Models, Formal Methods, Couplers, Verification

ACM Reference Format:
Chinmayi Baramashetru and Dominic Orchard. 2025. Towards Modelling and Verification of Coupler Behaviour in Climate Models. In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Programming for the Planet (PROPL '25), October 12–18, 2025, Singapore, Singapore.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3759536.3763801

PROPL '25, Singapore, Singapore

ACM ISBN 979-8-4007-2161-8/2025/10
https://doi.org/10.1145/3759536.3763801

1 Introduction

At the heart of efforts to understand our changing climate are *climate models*—large computational artefacts synthesising centuries of scientific understanding into multi-million line programs from which myriad potential future climatological scenarios are calculated. Like other large-scale software projects built by many people over decades, these models face significant engineering challenges. Whilst *validation* is often at the forefront of the scientific developer's mind (i.e., whether the model accurately represents the Earth system), *verification* remains a challenge (i.e., whether the program correctly implements the intended model) [22]. In safety-critical domains, formal verification is widely used, yet it is rare in climate modelling. Often, validation acts as a proxy for verification: if a model reproduces past climates or passes particular scientific 'benchmark' tests with acceptable error, then it is considered 'correct enough'. But when a model fails to produce expected results it is often unclear whether this is due to a failure of validity or verification [23]. In this work, we study large-scale climate models and consider whether there are inherent structural aspects that provide a fulcrum against which to lever formal verification techniques to aid modelling efforts by speeding up development, easing maintenance, and increasing trust. Ensuring coupler correctness is not an isolated concern; weak coupler guarantees can mask or propagate errors in component models, undermining the correctness of the entire coupled system. The models we consider are those used in the World Climate Research Programme's *Coupled Model Intercomparison Project*, which is a key data source for the *Intergovernmental Panel on Climate Change* (IPCC) reports. Such models split into *General Circulation Models* (GCMs) focused on the atmosphere and oceans, and *Earth System Models* (ESMs) which extend GCMs with models of the biosphere and/or cryosphere (e.g., sea ice, glaciers). Both types are often called 'coupled models' since they combine multiple models, usually the ocean and atmosphere at their core. This coupling not only connects two components but also models a real physical boundary between physical systems. From a programming perspective, coupling resembles function composition. However, ocean and atmosphere components typically differ in temporal and spatial resolutions, software architectures, and parallelisation strategies. Thus, independent pieces of software, or

Figure 1. Coupling structure in EC-Earth 3.1

libraries, called *couplers* are used to mediate between components, handling the field exchange and storage, grid remapping and interpolation, and time synchronisation. These couplers are complex, loosely specified orchestration layers. This lack of clear specification poses challenges for verifying correctness, reproducibility, and long-term maintenance, especially as models scale and run on heterogeneous hardware. Fortunately, couplers are often widely shared across models and research groups. Many of the 133 contributing models (and model configurations) in the 6th iteration of the Coupled Model Intercomparison Project (CMIP6 in 2021-22) use common coupler frameworks. This shared use makes couplers a high-impact target for verification that can have a large multiplicative effect.

We identify a set of key *principles* and *components* of couplers through a verification lens with the hope of shoring up the foundations of existing couplers and coupled models, and providing a route to future *verified coupler* efforts. This paper reports our initial analysis and outlines potential verification techniques, aiming to bridge the gap between the verification and climate modelling communities.

2 Coupled Models and Couplers

Software architecture has mainly focused on component-based software engineering to support modularity, maintainability, and reusability in complex systems [29]. *Couplers* (as introduced above) enable a component-based approach to climate modelling, reflecting different physical systems and different subdisciplines in model components ('submodels'). There has been growing interest in analysing the role and effect of coupler architectures on the scientific output of Earth System Models [2, 11]. The architectural patterns of various models reveal that couplers play a significant role in determining how tightly or loosely components are integrated, influencing the ease of adding new components and the fidelity of the simulated physical processes [2]. The choice of coupler logic can subtly alter climate feedbacks or introduce numerical biases, potentially affecting climate projections and scientific conclusions [20].

We examined commonly used couplers such as OASIS3-MCT [30, 31], CPL7 [10], and ESMF (the Earth System Modelling Framework) [14], identifying common core tasks, including: 1) transforming and regridding fields[1] between components, 2) managing data exchange and storage, 3) synchronising time steps across components with different time intervals, and 4) ensuring conservation of key physical quantities for physical consistency across components. Software errors in these coupler tasks, or in the model code surrounding coupling, can lead to subtle inconsistencies that may go unnoticed during standard testing and have unexpected effects on scientific results. The following illustrates how a lack of rigorous interface-level validation in a real-world model led to significant divergence in simulation outputs.

Example 2.1. In the EC-Earth3 ESM 3.1 model [12], freshwater runoff generated by the land model (H-TESSEL) is transferred to the ocean model (NEMO) via the OASIS3 coupler, illustrated in Figure 1. The coupler is responsible for spatial interpolation and routing fields between model components. Here, the OASIS coupler sends the runoff field from the land model to the ocean, but does not validate how it is received or how it is used by the ocean model. Massonnet et al. attempted to replicate this model by running ensemble simulations on two different HPC systems, showing significant differences in the ocean outputs [20]; the coupled model was sensitive to the change in computing platform. The divergence was traced to NEMO where a Fortran array in NEMO's runoff routine was not declared in the routine header, leading compilers to pre-fill it with different defaults. This meant that some elements of the receiving buffer retained arbitrary memory values unless overwritten. On one platform, the uninitialised entries appeared as zeros; on another, they appeared as NaNs. When combined with the valid runoff data, these spurious values entered the ocean surface, breaking ocean dynamics.

Although the root bug was in NEMO, the coupler played a role by transmitting to the field without checking its validity, allowing corrupted values to enter the coupled system unnoticed. Standard tests, such as bitwise comparison on a single platform or ensemble statistics, failed to detect the bug since the memory behaviour was consistent within one environment. The error only appeared across compilers. We argue that this bug could have been detected if the coupler had enforced explicit interface contracts, independent of compiler defaults. Contracts that require arrays to be initialised and to lie within physically valid bounds can detect invalid entries, such as negative values, NaNs, or values exceeding a maximum threshold. Similarly, postconditions that check for all-zero fields, or impose a tolerance condition on the total flux, can ensure that an array is not considered valid when it has been zero-filled by the allocator but not overwritten by the sender. In addition, conservation contracts can enforce that the total runoff transmitted by the land model equals

[1] In earth sciences, a 'field' is a function over space (and sometimes also time) representing a physical quantity. Numerical models approximate fields on a finite grid, i.e., as arrays of scalars indexed by co-ordinates in space/time.

(within some specified tolerance) the total received by the ocean model thereby catching cases where arrays are only partially updated. In EC-Earth3.2, OASIS3 was replaced by OASIS-MCT which includes additional checks on data. This highlighted the bug in NEMO, enabling it to be fixed by the developers. We thus advocate for more such checks but in a contract-based verification approach, providing a more systematic way to ensure correctness and consistency across model exchanges. This example strengthens the argument that coupler interface verification improves not only the coupler's robustness but also overall model correctness.

3 Towards Verifiable Coupler Design

Formal verification offers mathematically rigorous methods to ensure that software behaves as specified, catching errors beyond traditional testing [24]. The *Design by Contract* (DBC) [21] approach bounds software components by contracts that precisely define their expected behaviour through preconditions, postconditions, and invariants. Such formal contracts enable both static and runtime verification.

We bring this paradigm of verification to bear on the problem of coupler verification. In Section 3.1, we analyse a range of widely used, legacy, and modern couplers to identify their common core functional modules, refining the main categories of tasks identified in Section 2. Section 3.2 outlines key principles for a general coupler design, describes the role of each module and proposes an informal interface contract for each module in terms of preconditions, postconditions, and invariants. Finally, Section 3.3 gives a concrete instance of this approach, using the ACSL modelling language and a hybrid verification approach.

3.1 Major Couplers in Earth System Models

The OASIS family of couplers [8, 30, 32] is widely used in European climate models. OASIS3 is a configuration-driven coupler that exchanges two-dimensional fields between components. OASIS4 generalised the coupler to support three-dimensional fields and parallel communication, using a central server process to exchange data among multiple components. In contrast, OASIS-MCT (OASIS using the Model Coupling Toolkit, discussed further below) no longer runs as a separate central process; instead, each component model is linked directly to the coupler as a library. Through its API, the coupler provides functions for model initialisation, grid registration and remapping, and field declaration and transfer. Data exchanges between components are carried out in parallel using the Message Passing Interface (MPI). All coupling behaviour is externally configured. Key subsystems of the coupler include a field register, grid manager, interpolator, time manager, and a communication scheduler.

The Earth System Modeling Framework (ESMF) [14], developed by NASA, provides a component-based architecture with standardised interfaces. Each ESMF component has a standardised interface of methods (initialise, run, and finalise). ESMF allows components to be composed hierarchically, enabling a scalable architecture. Data is encapsulated and passed via a data structure containing arrays and numerical fields. ESMF's infrastructure layer includes field registries, grid classes (flexible subtypes for different physical grids), I/O classes for fields and grids, time management, and unified error handling. Its strong emphasis on formal interfaces resembles a lightweight Design-By-Contract approach where each component must provide specific fields at an agreed time and implement initialisation and finalisation methods.

The Model Coupling Toolkit (MCT) [17] is a lower-level library that has been integrated into frameworks such as OASIS, CPL6 and CPL7. The library provides essential functions such as data transfer and interpolation, but relies on higher-level frameworks for tasks like time synchronisation, metadata management, and initialisation.

CPL6 and CPL7 couplers developed at the US National Centre for Atmospheric Research (NCAR), as a community couplers [9, 10], connect all components to a single coupler component. Both CPL6 and CPL7 use MCT for data transfer and data interpolation. In CPL6, components call the coupler API directly, whereas in CPL7, it's built as a single executable with a single high-level driver– the main program that drives the overall execution sequence of all component models. The key subsystems in CPL6/7 include a grid module, with optional interpolation, initialisation, and time management with driver clocks and coupling frequency.

Other notable examples include the community couplers C-Coupler1 [18] and C-Coupler2 [19], which provide uniform runtime environments with similar modular subsystems to those already described.

Across all the couplers listed above, despite differences in their implementations, architectures and coupling strategies, a general set of core functional modules can be abstracted. The design principles for such modules provide a foundation for language-agnostic, modular coupler logic. This enables module-level contracts and verification strategies that can be systematically applied across different coupler implementations, which we exemplify in Section 3.3.

3.2 Coupler Subsystems and Functional Modules

We identify core functional modules of couplers and propose requirement specifications as potential preconditions, postconditions, and invariants as module-level contracts.

Configuration and Initialisation Module: parses external configuration or API input. For example, `namcouple` in OASIS3, which is a plain-text configuration file listing all fields, grids, interpolation methods, exchange frequency, and transformations. In ESMF, coupling parameters are specified through XML metadata and code-based settings. These configurations define which components are coupled, the frequency of data exchange, and the initialisation of internal

data structures. Similar configuration modules exist in other couplers, such as OASIS-MCT, C-Coupler, and CPL7.

Precondition: Configuration files or API calls must specify all coupling data, e.g., every field has a defined source and target, and each component must have grid information.

Postcondition: Each coupling component is defined, and the model explicitly specifies which fields it exchanges with other components. Supporting data structures, such as field registries and grid mappings, are also created at this stage.

Invariants: Every registered field must have a source, target, and unique identifier with no duplication.

Field Register and Metadata Module: manages individual coupling fields—in Earth sciences, a *field* refers to a physical quantity (e.g., temperature, salinity, runoff) represented as a function over space and sometimes time, which numerical models approximate as arrays of values on a grid. Each field is registered for exchange along with metadata such as unit, grid, and datatype. During initialisation, each component uses API calls to declare the fields it will send or receive. The coupler compiles these into a global registry that specifies, for each field, its source, target, mapping, and communication pattern. This registry acts as the implicit contract between components specifying their coupling behaviour.

Precondition: Components must declare fields together with the required metadata, including the field name, associated grid, data type, and whether the field is to be sent or received.

Postcondition: The coupler registry contains an entry for each coupler field with references or pointers to source and target components, field dimensions, and coupling timestep. For instance, after registration in ESMF, a field is part of the coupling contract, the coupler connector will expect the source to provide it and the target to accept it.

Invariants: Each field must have a unique identifier, every required field has a corresponding provider field, i.e., no dangling fields, and the grid size is immutable once the run starts.

Grid Manager and Interpolation Module: defines the grid for every component, generates interpolation weights and remaps the data from the source grid field to the target grid field. In some couplers, weight generation is offline (precomputed), while others compute weights in the initialisation module (see above).

Precondition: Each component grid must be defined and known to the coupler. Source and target grids for every exchange must be available and compatible with the interpolation method. Preconditions depend on the interpolation method, e.g., if using a conservative method, grid definitions must be available. Some couplers generate weights at runtime, which may need runtime assertion checks as contracts.

Postcondition: For every required source-target grid pair in the coupling, a mapping operator is defined, such as a sparse matrix in MCT or an analytical transformation. Given the source field and interpolation method, the coupler can produce the correct interpolated field on the target grid.

Invariants: If a conservation scheme is used, then the sum or integral of the field is conserved within the specified tolerance. The interpolation scheme should be consistent, ensuring a fixed mapping from source to target, unless the coupler supports an adaptive mesh strategy, which no mainstream couplers fully support currently.

Time Synchronisation Module: coordinates the coupling schedule between components and triggers data exchanges at defined intervals. In couplers with a single executable design, such as CPL7 and ESMF, the time advancement is handled by a central loop often called a 'coupler driver loop', which manages all component execution and data exchanges. In a multi-executable setup (e.g., OASIS) components synchronise their local time and data exchanges using explicit interface calls to the coupler.

Precondition: Each component has declared the coupling interval through a configuration file or time management interface. Clock or calendar conventions between components should be compatible.

Postcondition: At each step, the source produces data and the target consumes it at time `t`. A time manager enforces a synchronisation barrier so no component advances past `t` until the exchange completes. The corresponding precondition is simply that all components have reached time `t`.

Invariants: The time module must ensure that the simulation time never goes backwards. The coupler should not allow an exchange at time `t2` before the exchange at an earlier time `t1` (with `t1 < t2`). No coupling event should be duplicated or missed. The time module must ensure consistency between components, e.g., if two components exchange data every 3 hours, they should have the same clock-integration point (one cannot send every 3 hours while the other does it hourly); the time module must ensure consistent exchange intervals throughout.

Data Transfer and Routing Module: handles the actual sending and receiving of data fields between components. This module defines the communication patterns (MPI or shared memory) to exchange data fields. In a parallel routing context, the module uses various strategies such as MCT's router, OASIS3's gather-scatter approach, or intermediate grouping by exchanging via exchange grids to route the data.

Precondition: Before exchanging, the source component has valid data and the target component must be allocated and ready to accept data. The time synchronisation module has confirmed a valid coupling step, and the data field dimensions match those of the target.

Postcondition: Target fields are filled with data, and the target matches the source field or is transformed, following the postcondition of the interpolation module. For instance, in OASIS-MCT, a call to exchange data from an atmosphere

field to an ocean field results in the ocean receiving the atmosphere field, interpolated to the ocean's grid.

3.3 Example Formal Specification

In a deductive verification approach, formal specifications generate proof obligations that must hold for all possible executions. Several tools support deductive verification such as KeY [1] for Java using JML, SPARK [4] for Ada and Frama-C [16] for C using the ANSI/ISO C Specification Language (ACSL). Unfortunately, Fortran, the dominant language for climate models, has no such deductive tool (other than a small prototype in the CamFort tool [7]). We thus consider interoperating with Fortran code as a black box. However, postconditions involving native Fortran code cannot be verified statically because the verifier cannot symbolically analyse external compiled modules. Instead, we consider *runtime* verification. Runtime checks are similar to testing but driven by formal specifications, providing evidence of contract violations at runtime. Similar approaches have been successfully explored in safety-critical and high-assurance systems [5, 13]. This provides a balanced strategy for large, legacy scientific systems where full source-level verification is challenging. Although most climate models are primarily written in Fortran, C serves as the interoperability layer in high-performance scientific codes. Fortran codes routinely expose or call C bindings via the Fortran standard ISO_C_BINDING interface, and many core libraries in climate models (e.g., MPI, NetCDF) are themselves C-based. Hence, we leverage Frama-C, using the WP plugin for static verification and runtime assertion checking via the E-ACSL plugin [27]. ACSL contracts allow us to specify a high-level interface guarantees for coupler modules directly at the C layer, ensuring that verification integrates naturally with existing Fortran-to-C workflows.

ACSL contracts use various clauses to specify function behaviour. Preconditions (`requires`) describe the conditions that must hold when the function is called, while postconditions (`ensures`) describe the guarantees that must hold when the function terminates. A minimal schematic example is shown below:

```
/*@ requires P;
    ensures Q; */
return_type f(param_type ...);
```

Listing 1. Minimal ACSL function contract with requires and ensures clauses

Here, P and Q are logical predicates. In practice, these are expressed with built-in ACSL primitives such as asserting that a pointer is readable (`\valid_read(p)`), asserting that a pointer refers to allocated memory (`\valid(p)`), or referring to the return value (`\result`). The clause `assigns` specifies exactly which memory locations a function may modify, making data dependencies explicit and preventing hidden side effects. More complex user-defined predicates, such as

```
/*@ predicate bounds{L}(const double *a, integer n, real M)=
\forall integer i; 0 <= i < n ==> 0.0 <= a[i] <= M; @*/

/*@ predicate not_all_zero{L}(const double *a, integer n) =
    \exists integer i; 0 <= i < n && a[i] > 0.0; @*/

/*@ global invariant src_pos: SourceGrid  > 0;
    global invariant tgt_pos: TargetGrid  > 0;
    global invariant max_pos: MAX_VALUE   >= 0.0; @*/

/*@ requires n == SourceGrid;
    requires \valid_read(field + (0 .. n-1));
    requires bounds(field, n, MAX_VALUE);

    requires m == TargetGrid;
    requires \valid(out + (0 .. m-1));

    assigns out[0..m-1] \from field[0..n-1], n, m,
     ↪ MAX_VALUE;
    ensures bounds(out, m, MAX_VALUE);
    ensures not_all_zero(out, m); */
void remap_field(const double *field, int n, double *out,
     ↪ int m) { /* abstract model for remapping */ }

/*@ requires m == TargetGrid;
    requires \valid_read(remapped + (0 .. m-1));
    requires bounds(remapped, m, MAX_VALUE);

    requires \valid(routed + (0 .. m-1));

    assigns routed[0..m-1] \from remapped[0..m-1], m;
    ensures bounds(routed, m, MAX_VALUE);
    ensures not_all_zero(routed, m); */
void route_field(const double *remapped, int m, double
     ↪ *routed) { /* abstract model for routing */ }
```

Listing 2. Conceptual CouplerAdapter modules with ACSL

value ranges or conservation properties, can also be specified in the same way. In ACSL, a *global invariant* applies to specified global variables.

Our approach focuses on verifying core coupler modules at the interface level, rather than the numerical or physical kernels. We return to example 2.1 to showcase a contract-based verification strategy using ACSL specifications in Frama-C and the E-ACSL runtime verification framework. While OASIS-MCT is not modular internally, we propose to conceptually decompose its workflow (following Section 3.2) into stages such as initialisation, grid remapping, time synchronisation, and data transfer. In practice, the Fortran coupler wraps C adapter functions via ISO_C_BINDING; these adapters in turn call the OASIS C API and, when required, register C callbacks that can call back into Fortran routines also exposed through ISO_C_BINDING. The wrappers are treated as verification points where we attach explicit ACSL contracts. Since the EC-Earth 3.1 bug was due to the ocean

model using an uninitialised array to receive data from the land model, we add ACSL contracts to the modules to catch this in Listing 2. The code only shows the specification and not any surrounding additional code instrumentation. Here, `remap_field` abstracts interpolation and grid remapping in OASIS-MCT, with precomputed weights from `namcouple` and internal `oasis_get` and `oasis_put` calls) and the function `route_field` represents the final data transfer to the receiving model.

To address the uninitialised array bug, we introduce two reusable predicates. The first, `bounds`, to express that every array element lies in [0,MAX_VALUE]. This predicate also checks for NaN values as they fail ordered comparisons. The second, `not_all_zero`, is an existential condition that requires that at least one element of the array is strictly positive. Similarly, in cases where arrays are only partially overwritten or padded with zeros, a conservation contract can be applied to check that the total runoff flux sent by the land model equals the total received by the ocean model. For simplicity and to avoid additional code instrumentation in the example, the conservation contract check is omitted from the listing. The preconditions require (i) correct sizes for source/-target grids, (ii) valid memory, and (iii) values within physical bounds. The postconditions guarantee that the outputs also satisfy the bounds, ensuring that NaN or zero-filled data cannot pass through unnoticed. The `assigns` clauses make data dependencies explicit (e.g., `out` depends only on field, sizes, and MAX_VALUE; `routed` depends only on `remapped` and `m`), avoiding any hidden state. In addition, simple global invariants (SourceGrid >0, TargetGrid>0, MAX_VALUE ≥ 0) ensure valid assumptions globally.

Since these C wrappers call external Fortran code, post-conditions cannot be proved using Frama-C's WP plugin. We leverage the E-ACSL plugin to enforce the same contracts at runtime, detecting violations when they occur. One relevant comparison is with memory sanitiser tools such as Memcheck [26] and AddressSanitizer (ASAN) [25], which are widely used to detect memory errors. However, both are limited to low-level memory safety, and cannot express or enforce semantic correctness conditions across model interfaces. In contrast, E-ACSL enables runtime verification not only of pointer validity and array bounds, but also of user-defined conditions such as value ranges, conservation, or coupling synchronisation. While this may cause higher runtime overhead than ASAN depending on the contract complexity, E-ACSL integrates with Frama-C, enabling domain-specific correctness properties that cannot be checked by low-level tools. This hybrid verification strategy combines interface properties such as pointer validity, array bounds, and declared data dependencies at the C layer with runtime monitoring for black-box native code, providing complementary guarantees. Unlike reactive techniques such as output comparison, ensemble testing, or scattered runtime checks inside individual models, centralised contracts in the coupler can enforce cross-component checks and are auditable in a single location in code.

4 Conclusion

Formal verification techniques have helped safety-critical domains such as avionics [28] but remain largely unexplored in climate modelling. Prior work focuses on component-level validation via testing [6], numerical verification [15], model checking [3, 5], and the software architecture of couplers [2], but no work addresses cross-component verification at the level of couplers. We aim to bridge this gap, showing how lightweight formal methods can employ post-hoc verification, ensure safety guarantees, and support interoperability across coupler components. We extract coupler design principles and propose verifiable modules using ACSL contracts, verified with Frama-C and E-ACSL. We plan to prototype this approach by wrapping and verifying the OASIS-MCT coupler and retrofitting our strategy to existing couplers. This technique could also be applied in the construction of a new verified coupler, and even in couplers written in dynamic languages such as Julia or Python, which offer runtime flexibility and could benefit from lightweight contract verification. By introducing formal interface contracts into coupler design, we move towards auditable coupled models, systems whose correctness is not only empirically validated but also formally verifiable.

Acknowledgments

Thank you to the reviewers for their comments which helped to improve this paper in its final form. This research received support through Schmidt Sciences, LLC. Thank you also to the Institute of Computing for Climate Science for enabling this work through its supportive research environment.

References

[1] Wolfgang Ahrendt, Thomas Baar, Bernhard Beckert, Richard Bubel, Martin Giese, Reiner Hähnle, Wolfram Menzel, Wojciech Mostowski, Andreas Roth, Steffen Schlager, et al. 2005. The KeY tool: integrating object oriented design and formal verification. *Software & Systems Modeling* 4 (2005), 32–54. https://doi.org/10.1007/s10270-004-0058-x

[2] Kaitlin Alexander and Stephen M Easterbrook. 2015. The software architecture of climate models: a graphical comparison of CMIP5 and EMICAR5 configurations. *Geoscientific Model Development* 8, 4 (2015), 1221–1232. https://doi.org/10.5194/gmd-8-1221-2015

[3] Alper Altuntas, Allison H Baker, John Baugh, Ganesh Gopalakrishnan, and Stephen Siegel. 2025. Specification and Verification for Climate Modeling: Formalization Leading to Impactful Tooling. https://doi.org/10.4018/9781599042190.ch005 VSS 2025: International Workshop on Verification of Scientific Software.

[4] John Gilbert Presslie Barnes. 2003. *High integrity software: the spark approach to safety and security: sample chapters.* Pearson Education.

[5] John Baugh and Alper Altuntas. 2018. Formal methods and finite element analysis of hurricane storm surge: A case study in software verification. *Science of Computer Programming* 158 (2018), 100–121. https://doi.org/10.1016/j.scico.2017.08.012

[6] Tom Clune, Natalie Patten, Ben Auer, and Arlindo da Silva. 2023. Component Level Testing in a Hierarchical Architecture. In *Workshop on*

Correctness and Reproducibility for Climate and Weather Software.

[7] Mistral Contrastin, Matthew Danish, Dominic Orchard, and Andrew Rice. 2016. Lightning Talk: Supporting Software Sustainability with Lightweight Specifications. In *Proceedings of the Fourth Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE4), University of Manchester, Manchester, UK, September 12-14*, Vol. 1686. CEUR Workshop Proceedings.

[8] Anthony Craig, Sophie Valcke, and Laure Coquart. 2017. Development and performance of a new version of the OASIS coupler, OASIS3-MCT_3. 0. *Geoscientific Model Development* 10, 9 (2017), 3297–3308. https://doi.org/10.5194/gmd-10-3297-2017

[9] Anthony P Craig, Robert Jacob, Brian Kauffman, Tom Bettge, Jay Larson, Everest Ong, Chris Ding, and Yun He. 2005. CPL6: The new extensible, high performance parallel coupler for the Community Climate System Model. *The International Journal of High Performance Computing Applications* 19, 3 (2005), 309–327. https://doi.org/10.1177/1094342005056117

[10] Anthony P Craig, Mariana Vertenstein, and Robert Jacob. 2012. A new flexible coupler for earth system modeling developed for CCSM4 and CESM1. *The International Journal of High Performance Computing Applications* 26, 1 (2012), 31–42. https://doi.org/10.1177/1094342011428141

[11] Robert E Dickinson, Stephen E Zebiak, Jeffrey L Anderson, Maurice L Blackmon, Cecelia De Luca, Timothy F Hogan, Mark Iredell, Ming Ji, Ricky B Rood, Max J Suarez, et al. 2002. How can we advance our weather and climate models as a community? *Bulletin of the American Meteorological Society* 83, 3 (2002), 431–436. https://doi.org/10.1175/1520-0477(2002)083<0431:hcwaow>2.3.co;2

[12] Ralf Döscher, Mario Acosta, Andrea Alessandri, Peter Anthoni, Almut Arneth, Thomas Arsouze, Tommi Bergmann, Raffaele Bernadello, Souhail Bousetta, Louis-Philippe Caron, et al. 2021. The EC-earth3 Earth system model for the climate model intercomparison project 6. *Geoscientific Model Development Discussions* 2021 (2021), 1–90. https://doi.org/10.5194/gmd-15-2973-2022

[13] John Hatcliff, Gary T Leavens, K Rustan M Leino, Peter Müller, and Matthew Parkinson. 2012. Behavioral interface specification languages. *ACM Computing Surveys (CSUR)* 44, 3 (2012), 1–58. https://doi.org/10.1145/2187671.2187678

[14] Chris Hill, Cecelia DeLuca, Max Suarez, ARLINDO Da Silva, et al. 2004. The architecture of the Earth System Modeling Framework. *Computing in Science & Engineering* 6, 1 (2004), 18–28. https://doi.org/10.1109/MCISE.2004.1255817

[15] Franjo Ivančić, Malay K Ganai, Sriram Sankaranarayanan, and Aarti Gupta. 2010. Numerical stability analysis of floating-point computations using software model checking. In *Eighth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE 2010)*. IEEE, 49–58. https://doi.org/10.1109/memcod.2010.5558622

[16] Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, and Boris Yakobowski. 2015. Frama-C: A software analysis perspective. *Formal aspects of computing* 27, 3 (2015), 573–609. https://doi.org/10.1007/s00165-014-0326-7

[17] Jay Larson, Robert Jacob, and Everest Ong. 2005. The model coupling toolkit: A new Fortran90 toolkit for building multiphysics parallel coupled models. *The International Journal of High Performance Computing Applications* 19, 3 (2005), 277–292. https://doi.org/10.1177/1094342005056115

[18] Li Liu, G Yang, B Wang, C Zhang, R Li, Z Zhang, Y Ji, and L Wang. 2014. C-Coupler1: A Chinese community coupler for Earth system modeling. *Geoscientific Model Development* 7, 5 (2014), 2281–2302. https://doi.org/10.5194/gmd-7-2281-2014

[19] Li Liu, Cheng Zhang, Ruizhe Li, Bin Wang, and Guangwen Yang. 2018. C-Coupler2: a flexible and user-friendly community coupler for model coupling and nesting. *Geoscientific Model Development* 11, 9 (2018), 3557–3586. https://doi.org/10.5194/gmd-11-3557-2018

[20] François Massonnet, Martin Ménégoz, Mario Acosta, Xavier Yepes-Arbós, Eleftheria Exarchou, and Francisco J Doblas-Reyes. 2020. Replicability of the EC-Earth3 Earth system model under a change in computing environment. *Geoscientific Model Development* 13, 3 (2020), 1165–1178. https://doi.org/10.5194/gmd-13-1165-2020

[21] Bertrand Meyer. 2002. Applying'design by contract'. *Computer* 25, 10 (2002), 40–51. https://doi.org/10.1109/2.161279

[22] William L Oberkampf and Christopher J Roy. 2010. *Verification and validation in scientific computing*. Cambridge university press.

[23] Dominic Orchard and Andrew Rice. 2014. A computational science agenda for programming language research. *Procedia Computer Science* 29 (2014), 713–727. https://doi.org/10.1016/j.procs.2014.05.064

[24] John Rushby. 1993. *Formal methods and the certification of critical systems*. Vol. 37. SRI International, Computer Science Laboratory. https://doi.org/10.1007/978-1-4471-0921-1_1

[25] Konstantin Serebryany, Derek Bruening, Alexander Potapenko, and Dmitriy Vyukov. 2012. {AddressSanitizer}: A fast address sanity checker. In *2012 USENIX annual technical conference (USENIX ATC 12)*. 309–318. https://www.usenix.org/conference/atc12/technical-sessions/presentation/serebryany

[26] Julian Seward and Nicholas Nethercote. 2005. Using Valgrind to Detect Undefined Value Errors with Bit-Precision. In *USENIX Annual Technical Conference, General Track*. 17–30.

[27] Julien Signoles, Nikolai Kosmatov, and Kostyantyn Vorobyov. 2017. E-ACSL, a runtime verification tool for safety and security of C programs (tool paper). In *RV-CuBES 2017-International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools*. https://doi.org/10.29007/fpdh

[28] Jean Souyris, Virginie Wiels, David Delmas, and Hervé Delseny. 2009. Formal verification of avionics software products. In *International symposium on formal methods*. Springer, 532–546. https://doi.org/10.1007/978-3-642-05089-3_34

[29] Clemens Szyperski, Dominik Gruntz, and Stephan Murer. 2002. *Component software: beyond object-oriented programming*. Pearson Education. https://doi.org/10.5381/jot.2005.4.3.a3

[30] S Valcke. 2013. The OASIS3 coupler: A European climate modelling community software. *Geoscientific Model Development* 6, 2 (2013), 373–388. https://doi.org/10.5194/gmd-6-373-2013

[31] Sophie Valcke, Tony Craig, and Laure Coquart. 2013. OASIS3-MCT user guide, oasis3-mct 2.0. *CERFACS/CNRS SUC URA* 1875 (2013).

[32] Sophie Valcke, René Redler, and Reinhard Budich. 2011. *Earth system modelling-volume 3: Coupling software and strategies*. Springer Science & Business Media. https://doi.org/10.1007/978-3-642-23360-9