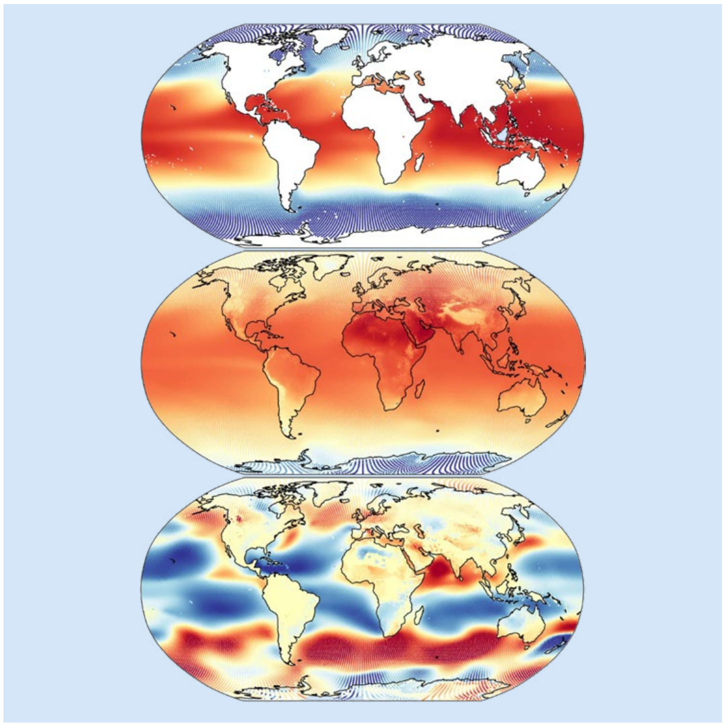
ECMWF Feature article

from Newsletter Number 182 – Winter 2024/25

COMPUTING

Modernisation of the Integrated Forecasting System



www.ecmwf.int/en/about/media-centre/media-resources

doi: 10.21957/m9ad5hv72s

This article appeared in the Computing section of ECMWF Newsletter No. 182 – Winter 2024/25, pp. 19–23

Modernisation of the Integrated Forecasting System

Michael Sleigh, Andrew Bennet, Paul Burton, Paul Cresswell, Patrick Gillies, Adrian Hill, Zak Kipling, Michael Lange, Olivier Marsden, Ahmad Nawab, Balthasar Reuter

The Integrated Forecasting System (IFS) is mission-critical software for ECMWF. It fulfils our primary purposes of (a) developing a capability for medium-range weather forecasting, and (b) providing medium-range weather forecasts to our Member and Co-operating States, and it is also used for other applications. The IFS is, however, also extremely complex. The main driver in its development has been the need to improve meteorological quality and capability – an effort to which hundreds of people have contributed over decades. While very successful in terms of providing forecasts, maintaining and updating the IFS has also led to the accumulation of a great deal of technical debt in the system. Now, because of growing demands on the system and increasing diversity of the environment, the rate of accumulation of technical debt is increasing. This leads to the need to modernise the IFS by adopting a modular design, a new representation of data, and an open-source approach.

The need to modernise

The growing demands on the system come from the expanding range of applications it is used in, and the growing population of developers and researchers who work with it. Since its launch, the IFS has evolved to support various configurations and applications: 4D-Var data assimilation, ensemble forecasting; ensemble data assimilation; full Earth system modelling; sub-seasonal and seasonal forecasting; atmosphere, ocean and land reanalysis; atmospheric composition and greenhouse gas forecasting and reanalysis; flood and fire forecasting; use in academic research/teaching (via OpenIFS); and most recently, with Destination Earth (Geenen et al., 2024), in climate and extreme weather at particularly high resolution.

Additionally, the environment in which it operates is becoming more complex and challenging. For a long time, all our operational and research work ran in an ECMWF-owned data centre and high-performance computing (HPC) facility, and it was able to use an industry-standard combination of central processing units (CPUs) and Message Passing Interface–Open Multi-Processing (MPI–OpenMP) parallelism. The first of these has ceased to be true, with Destination Earth pioneering the move to take advantage of external data centres and HPC machines we do not administer. On the second point, much effort has already been made to adapt to graphics processing units (GPUs), over several years. These developments together presage a much more diverse and fast-changing computing environment, which will include off-premises and cloud components and a wider range of vendors. The range of architectures has already broadened, and although we might not expect rapid change in the hardware market in the foreseeable future, at the very least it will become essential to utilise equally well those architectures that have already emerged. The rise of successful machine learning methods in weather forecasting also creates many opportunities for use in a complex, hybrid operational system, with the additional challenges that brings. As noted in the ECMWF Strategy from 2025 to 2034, a high degree of flexibility and agility is called for, to be able to respond to rapid changes. The increasing need to be open in our science, data and software applies additional pressure. Furthermore, it will become more difficult to find software engineers who are willing to work on 'old' software frameworks and computing languages, such as Fortran.

The above factors – prioritising progress in performance and capability, continual increases in demands on the system due to diversifying applications, and a rapid increase in the complexity of the environment – have led to a system which is difficult to work with: less easy to understand, more brittle (easy to break), harder to test, larger and more complex than is necessary, and hence slower to improve. Like any debt that continually increases, there is a risk that at some point this becomes unsustainable. This might happen because of increasing complexity, gradually eroding our ability to make functional improvements in a realistic amount of time, at a time when the need to move quickly is at a premium.

This challenge points to an urgent need to modernise the architecture and infrastructure on which our forecasting systems are based; to clear much of the legacy of technical debt already inherent in the system; and to implement for the long term a development approach in which technical debt is explicitly recognised, documented, and dealt with continuously, alongside functional improvements. Much effort and progress have been made in recent years, for example through the Scalability Programme (Bauer et al., 2020); a project to adapt the IFS so that it is ready for a hybrid CPU–GPU compute model (Hybrid 2024); and continual modernisation of our infrastructure and research-to-operations (R2O) tools and processes (Buizza et al., 2017; Buizza et al., 2018). But a more concerted, coordinated and exhaustive strategy is needed.

The necessary improvements can be achieved by the application of familiar and well understood software engineering techniques and processes to the design and the ongoing development of our forecasting-system software. The central part of this strategy is therefore the encapsulation, where appropriate, of compact, standalone components that can be individually tested and developed, at least to some extent, in isolation from the whole system, i.e. the separation of concerns. Efforts along these lines are not wholly new: many of them were implemented through the Scalability Programme, which introduced concepts of modularisation and separation of concerns in the form of an overarching plan to adapt the IFS to forthcoming HPC architectures. Hence, much of what we propose here builds on these initial successes and implements an overarching approach that takes the separation of concerns in a more coordinated manner to its logical conclusion.

We also propose additional actions that complement this approach, such as the definition of API (application programming interface) specifications and rigorous versioning. These ensure that integrated systems can be composed from the encapsulated components without descending into ‘dependency hell’. New and more extensive standards will be introduced to guide developers, and tooling will be developed initially to detect and fix departures from the standards, and later to automatically enforce them. Standards will be of particular importance because, in addition to guiding developers, they will ensure the code is suitably structured to allow in-house tools to operate on it. An example of such a tool is Loki, which is an in-house source-to-source code translation tool. This will also facilitate the automatic extraction of OpenIFS, a supported and easily accessible version of the IFS provided for research and education, from the larger IFS source code. Standards and tools will also recognise the importance of code deprecation and removal.

Importantly, the strategy of devolving the code into components is not to move away from the idea of being integrated: we still intend to meet the same range of applications from a single source code base. Rather, it is to move away from an integrated forecasting system that is a monolith, to one that is a coherent ‘ecosystem’ of components from which forecast applications can be composed.

Component design

The overarching principle is the move towards a truly modular overall design, in contrast to the traditional monolithic approach that was used to create the core IFS code. Importantly, there will be a separation between technical infrastructure, individual scientific components, and the different variations of the overarching codes. This will ensure that technical and scientific changes can be adopted and migrated easily between the various supported cycles and configurations of the IFS. Model components are shared with Météo-France and the ACCORD consortium, and careful consultation will be undertaken to determine how best to separate code into a set of self-contained libraries that meets everyone’s requirements. Additional testing infrastructure will be deployed to better track significant scientific and technical changes and coordinate the respective inclusion in release cycles across different organisations and projects.

A modular component structure makes it possible to improve test coverage and streamline change management. Individual module-level testing will facilitate greater test coverage. It will also enable the testing of non-operational features and model-specific code paths in jointly developed core modules, in addition to rigorous technical testing of software infrastructure components. Importantly, this does not diminish our current strong emphasis on scientific testing and evaluation in the R2O process. Here, components need to be integrated and tested together to understand their cumulative impact, no matter how well tested any individual component is in advance. We propose that the frequency of scientific release cycles will not be affected (with one major cycle per year on a fixed schedule), while their coordination will be improved.

A key change that such a refactoring will bring is that code ownership and governance can be applied per component. While scientific and infrastructure packages that include jointly developed code will require agreement and approval from all stakeholders, other components will be under local ownership. This does not preclude external contributions or use of specific subroutines in other modules. However, it makes the testing and technical management responsibility as localised as possible, with external contributions going through formal code review and integration testing before cycle synchronisation.

Data structures

A prime consideration in a more modular design is the representation of data, in particular gridded field data in grid-point and spectral space representations. Many of the modular components that will be developed in the approach described above will benefit from a harmonised representation of such field data. This will be provided by a new standalone library to enhance the modularity and uniformity of component interfaces.

A central representation of field data structures will also enable a clean separation between scientific abstractions used in forecast and data assimilation contexts, and technical concepts. The latter include concepts such as offload to accelerators and programming model compatibility. This separation can happen along the following lines:

- **Scientific API:** Here, further use of object-oriented Fortran and inheritance can separate high-level concepts, for example prognostic and diagnostic variables, and facilitate quick access to scientific metadata. Similarly, encapsulating higher-level concepts, such as tendencies or surface variable groups, and consolidating their use across the code base will be done in close collaboration with scientists.
- **Computational API:** Existing features in Atlas, a software library supporting the development of Earth system model components and data processing (Deconinck, 2018), and other libraries will be expanded to generally encapsulate technical features such as data storage backends, GPU-compatibility, parallel communication support, and I/O interface and utilities.

Moreover, a uniform, object-oriented representation of field data will serve two purposes:

- Provide a common API for field abstractions, compatible with Atlas, that encapsulates technical, computational and scientific functionality.
- Store IFS forecast states (model state, grid and fields) in unified data structures that provide a clean interface to OOPS (a framework for running different variational data assimilation formulations with a variety of forecast models) for forecast and data assimilation configurations.

A unified and individually tested data structure library will enable the graceful migration towards operational use of Atlas as the default memory data backend. The use of Atlas in the IFS will allow wider compatibility with novel packages, such as the FVM (a new nonhydrostatic dynamical core under development at ECMWF).

Workflow code

Separate to the compiled executables, the surrounding ‘workflow code’ divides into: (1) scripts which drive the model and manage data flow at run-time, and (2) ‘suite builder’ code, which runs once at deployment to generate the structure of the suite.

The envisaged workflow architecture is summarised in Figure 1. Multiple run-time configurations are combined with a suite generator tool which builds an ecFlow suite definition. ecFlow is a work flow package that enables users to run a large number of programs in a controlled environment (see *https://ecflow.readthedocs.io*). The suite generator tool also builds shell wrappers for its tasks, and it initiates a deployment of those, alongside required static component configurations and dependent code libraries. Control is then passed to the ecFlow suite, which orchestrates the execution of the task wrappers, which optionally generate on-the-fly state-dependent configurations and invoke relevant binaries and library scripts.

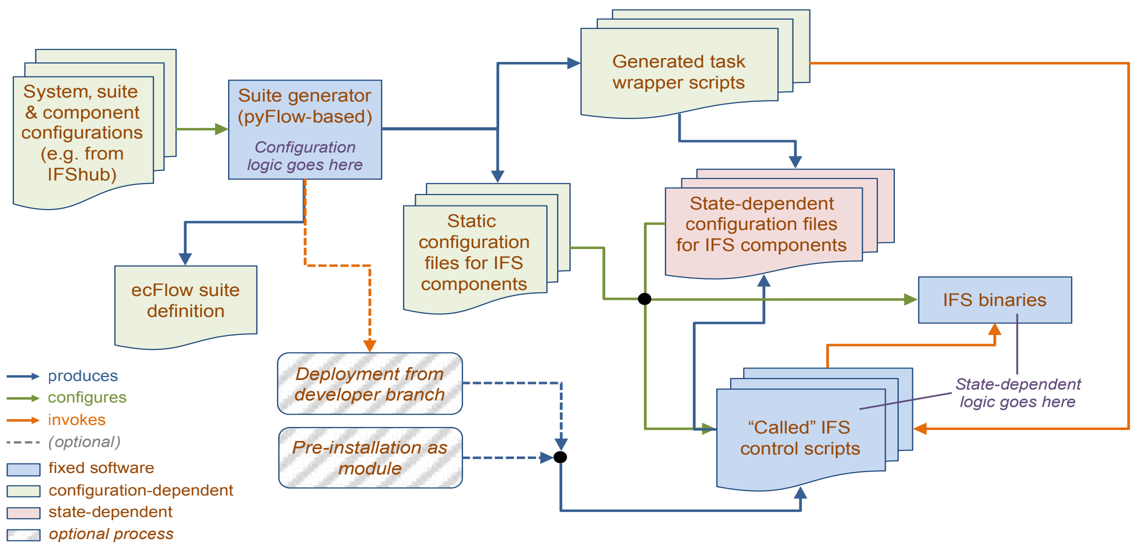

System, suite & component configurations (e.g. from IFShub)
Suite generator (pyFlow-based)
Configuration logic goes here
Generated task wrapper scripts
Static configuration files for IFS components
State-dependent configuration files for IFS components
IFS binaries
ecFlow suite definition
Deployment from developer branch
Pre-installation as module
"Called" IFS control scripts
State-dependent logic goes here
produces
configures
invokes
(optional)
fixed software
configuration-dependent
state-dependent
optional process

Figure 1 Future architecture of IFS workflow code, illustrating the interaction between suite generation, control scripts, configuration files, etc. Workflows are defined by the user in IFShub, a web-based interface for ECMWF and external users. Then, suite-generation code written in Python, and using the pyFlow library, generates both the ecFlow suite definition that defines the overall workflow (the set of tasks and how they interrelate and depend on each other), and the task wrapper scripts that control each task. Ultimately this leads to the production of forecasts via the IFS binaries. Care is needed to separate properly the static configuration defined once for the overall workflow, and the state-dependent configuration that evolves as the work proceeds.

There are currently multiple sets of code for generating and deploying suites. All are written in Python, but they have substantial differences in both functionality and style, with varying levels of use of common frameworks. None has the required flexibility to cover all the needs of the complete range of research and operational forecasting suites. The strategy is to build a system which delivers the necessary flexibility. This is to be done by unifying the design and implementation of suites across applications, based on a common software stack such as pyFlow and Troika. Current differences in the use of suites should be harmonised as much as possible. It is also crucial to harmonise between research and operational environments. Differences here have often been the source of unexpected issues during the R2O process.

To begin, ECMWF is developing a set of common standards for suite design. Both the tooling for suite generation, and the design of the suites themselves, will be developed in line with these standards as they evolve. Suite design standards and their publication as part of invitations to tender (ITTs) will ensure consistency and optimal design for both internal and contracted-out work. The introduction of a full end-to-end forecast-suite-level test platform, which we refer to as the 'development suite' (d-suite), will demonstrate how new workflow standards and designs will work in an operational-like context.

All forecast systems are driven at runtime by scripts, which have grown organically to a high level of complexity, with a variety of different styles and approaches. This leads to a significant maintenance burden and difficulty in making changes and adding new features. It also adds substantial overhead to the migration to other HPC platforms.

A major effort to restructure and refactor the scripts is proposed, with the aim of making them cleaner, more robust and maintainable, and easier to understand and modify with confidence. The starting point for this will be rolling out a recently published IFS shell scripting standard, in three stages:

- Migrating away from the legacy Korn shell to Bash.
- Implementing a clean separation between ecFlow task wrappers, calling plain shell scripts to do the actual work which can be tested outside of ecFlow.
- Bringing scripts into line with the detailed provisions in the standard.

In parallel with this, a longer-term effort will be to modularise scripts, factoring out common repeated code and breaking down large scripts into smaller pieces that can be meaningfully tested in isolation. Where appropriate, functionality should be factored out into self-contained tools (e.g. Python packages) to be installed on the target platform. The primary motivation for this encapsulation is that it results in a collection of components that can be tested in isolation, outside the context of an ecFlow suite.

Adoption of open source and open development

IFS software is partially open-source, following a Council agreement to open-source selected components while keeping the system as a whole closed. The process of open-sourcing components aims to enhance collaboration with both Earth system and computer scientists. To date, a number of scientific kernels, such as the radiation package ecRad, and technical infrastructure libraries have been made public under the Apache 2.0 licence.

The short-term plan is to include these existing open-source components in the operational IFS build, and to remove the corresponding code from the central IFS source repository, so all scientists and developers will work directly in the public repositories. This will benefit the modernisation of the code.

In the longer term, the new ECMWF Strategy says that "ECMWF will build on the successful OpenIFS efforts and move to an open-source approach for the whole of the forecast model." Council decisions permitting, an initial open-source code will be based on OpenIFS, along with all the infrastructure to permit running 'out-of-the-box'. Since the initial open-source version of the IFS forecast model will effectively be OpenIFS, we suggest that the new public offering retain for the long term the OpenIFS identity.

Testing

Testing is of critical importance. Most recent improvements in IFS development workflows at ECMWF have been improvements in testing. These have included introducing a fast interactive test framework at the compiled IFS code level, and continuous integration (CI) testing which works at the full integrated-suite level. Building on this, much of the motivation for the separation of concerns/abstraction/modularity made above is that it enables more, earlier, and faster testing. The full integration-level and scientific testing in our current R2O process is still required, but there is more that can be done earlier in the process.

One limitation in current technical testing is that only the testing of changes expected to be bit-identical has been automated. We have no procedures to automatically test and accept meteorologically neutral technical changes, such as those related to GPU adaptation, optimisation, or code refactoring, without recourse to running long experiments of the order of a month and making a manual assessment. This is a particular barrier for purely technical code developers, and for HPC vendors, who are likely to need to make technical changes to tune the IFS to their systems as part of any procurement benchmarking. One approach already used by MeteoSwiss is a test ensemble, in which confidence intervals for output values are determined using a known good configuration. These are then used to detect problems in runs on different setups (different compiler, optimisations, accelerators) (see *https://github.com/MeteoSwiss/probtest*). We will explore this and similar ideas for the IFS.

It is a critical requirement to be able to identically replicate behaviour and results of operations in a sandboxed test environment. We will ensure the full software environment of an IFS run is controlled and replicable between research and operations. Being able to run the same code and configuration in both research experiments and operational suites helps ensure we fully test changes only once, and early in their development.

In addition to improved integration-level testing, we plan to significantly expand the scope of unit testing to verify the behaviour of individual components. This applies at both the coarse level of the modular components (for example the ECMWF ocean wave model, ecWAM), but also where possible to individual routines and scripts. Such an approach is already taken for suite generation code, but it will be extended to scripts and compiled code.

As previously mentioned, the introduction of a 'd-suite' (a fully functional, end-to-end clone of the operational suite) as an intermediate integration testing system between forecast-system development and operations will be a key mechanism to support enhanced testing, harmonise between research and development and operations, and allow us to move towards continuous delivery.

Conclusion and outlook

We have proposed a software strategy for our forecasting system, to complement the wider ECMWF Software Strategy (Quintino et al., 2023). This is intended to meet the immediate-to-medium-term challenge of ensuring the forecasting system that we have now – the IFS – remains sustainable on at least a ten-year timescale. It intends to make our forecasting software substantially more agile to meet the challenges of a rapidly changing system and environment. An implementation project called FORGE (Forecast-System Regeneration) is being launched to deliver the strategy.

While the more immediate software challenges have been thought through in detail so far, another strand of software strategy, which is less mature at this stage, looks towards and beyond the ten-year horizon. In particular, this concerns how the forecasting system might move wholly away from the current IFS and Fortran to a new FVM-based system written in Python, and exploiting domain-specific languages (DSL) to separate scientific from technical concerns. This will be the subject of future articles.

Feedback is actively being sought from developers and researchers within ECMWF and our Member and Co-operating States, and from our collaborators and other stakeholders, on the detail of the approaches presented here. This is done to take account of the feedback in the ongoing process of developing a final detailed strategy document, covering both the more immediate software challenges and the longer term, which will be presented to ECMWF's Committees in the future.

Further reading

Bauer, **P.**, **T. Quintino**, **N. Wedi**, **A. Bonanni**, **M. Chrust**, **W. Deconinck et al.**, 2020: The ECMWF Scalability Programme: progress and plans. *ECMWF Technical Memorandum* **No. 857**. *https://doi.org/10.21957/gdit22ulm*

Bonavita, **M.**, **Y. Trémolet**, **E. Hólm**, **S. Lang**, **M. Chrust**, **M. Janiskova et al.**, 2017: A Strategy for Data Assimilation. *ECMWF Technical Memorandum* **No. 800**. *https://doi.org/10.21957/tx1epjd2p*

Buizza, **R.**, **E. Andersson**, **R. Forbes** & **M. Sleigh**, 2017: The ECMWF research to operations (R2O) process. *ECMWF Technical Memorandum* **No. 806**. *https://doi.org/10.21957/m3r9bvg6x*

Buizza, **R.**, **M. Alonso-Balmaseda**, **A. Brown**, **S.J. English**, **R. Forbes**, **A. Geer**, 2018: The development and evaluation process followed at ECMWF to upgrade the Integrated Forecasting System (IFS). *ECMWF Technical Memorandum* **No. 829**. *https://doi.org/10.21957/xzopnhty9*

Burton, **P.**, **M. Martins**, **S. Siemen** & **M. Sleigh**, 2021: IFShub: a new way to work with IFS experiments. *ECMWF Newsletter* **No. 167**, 28–32. *https://doi.org/10.21957/bu599oxq27*

Deconinck, **W.**, 2018: ECMWF releases Atlas software library. *ECMWF Newsletter* **No. 155**, 12–13. *https://www.ecmwf.int/en/newsletter/155/news/ecmwf-releases-atlas-software-library*

Geenen, **T.**, **N. Wedi**, **S. Milinski**, **I. Hadade**, **B. Reuter**, **S. Smart et al.**, 2024: Digital twins, the journey of an operational weather system into the heart of Destination Earth. *Procedia Computer Science*, **240**, 99–108. *https://doi.org/10.1016/j.procs.2024.07.013*

Quintino, **T.**, **U. Modigliani**, **F. Pappenberger**, **S. Lamy-Thepaut**, **S. Smart**, **J. Hawkes et al.**, 2023: Software strategy and roadmap 2023-2027. *ECMWF Technical Memorandum* **No. 904**. *https://doi.org/10.21957/c6d7df0322*