Microservices Architecture in Environmental Modelling: Potential and Feasibility

Qianhui Lin[1,2], Sam Harrison[1*], Faiza Samreen[1], Abdessalam Elhabbash[2], Gordon S. Blair[1]
[1]UK Centre for Ecology & Hydrology, Lancaster, United Kingdom
Emails: qialin@ceh.ac.uk, sharrison@ceh.ac.uk, FaiSam@ceh.ac.uk, gblair@ceh.ac.uk
[2]School of Computing and Communications, Lancaster University, Lancaster, United Kingdom
Emails: q.lin9@lancaster.ac.uk, a.elhabbash@lancaster.ac.uk
[*]Corresponding author

***Abstract*—Environmental models are crucial for understanding and predicting the behaviour of natural systems and can aid decision-making. However, many environmental models have traditionally been built as large monolithic systems, which makes them difficult to maintain, scale, and collaboratively develop. At the same time, there is growing recognition within the environmental modelling community of the value of adopting modern software architecture practices. Here, we investigate how such practices can support the development of environmental models, with a particular focus on microservices architecture (MSA). We first identified key challenges in environmental modelling and then reviewd existing efforts within the community, focusing on both their limitations and alignment with MSA principles. To demonstrate the potential benefits of applying microservices to environmental modelling, we refactored an existing monolithic model, which simulates the dispersion of contaminants in soil due to bioturbation by earthworms, into a microservice-based model. Our results show that adopting a MSA can improve system performance in terms of response time and throughput, while also enhancing scalability and resource efficiency. We also observed qualitative benefits, including better maintainability and support for collaboration when developing environmental models across teams. Overall, our study suggests that the MSA holds significant promise for environmental modelling and points to new directions for the development of future environmental models.**

***Index Terms*—microservices architecture, environmental modelling, monolithic architecture**

I. Introduction

Environmental models play a crucial role in understanding natural processes, forecasting outcomes, and supporting decision making. Traditionally, modelling efforts have focused on achieving high quality outputs and ensuring that models are both useful and reliable [1]. Yet, as environmental challenges become increasingly complex, there is a growing need for models that are not only scientifically accurate but also designed in ways that support long-term development, adaptation to new scenarios, and collaboration among different users and research groups. Within the environmental modelling community, several recurring challenges have been noted in achieving these goals.

First, maintaining and evolving existing models presents difficulties. Re-purposing existing model solutions for use in new scenarios can be difficult, especially with legacy codebases [2]. Many models are initially built to address specific research questions, with little thought for future expansion or collaborative development [2]. When later shared within the community, their limited extensibility makes maintenance and adaptation hard. In some cases, this even necessitates complete rewrites to incorporate new features or meet changing needs [2]. Furthermore, insufficient attention to maintainable and robust coding practices has resulted in low readability and further complicate refactoring [2].

Second, technology heterogeneity also creates challenges. Existing environmental models are implemented in a variety of programming languages; many legacy models were written in Fortran or C, while high-level languages such as Python and R are becoming increasingly popular in the community [3]. This heterogeneity creates challenges during model reuse or exchange, as integrating or coupling models written in different languages has been identified as a significant issue [4].

Third, scalability is an important consideration in environmental modelling. [2]. As models increase in complexity, such as incorporating larger datasets, finer spatial and temporal resolutions, and more intricate processes, hardware limitations can quickly become a bottleneck [5]. Additionally, load imbalance is an important concern; when computational workloads are unevenly distributed across workers (such as CPUs or computing nodes), it can lead to significant inefficiencies and reduced overall performance [5].

Fourth, collaborative development can be difficult to sustain in environmental modelling. Supporting ongoing development in environmental modelling requires processes that allow multiple contributors to update, refine, and extend models over time [6]. Although iterative development approaches and automated workflows are well established in other software development domains, they have been less consistently adopted by the environmental modelling community [7]. The lack of shared practices, such as continuous integration, version control conventions, and automated testing, can make it difficult for research groups to coordinate contributions and maintain reliability when multiple people work on the same codebase [2]. As a result, collaborative development often becomes slow, fragmented, and prone to errors, limiting the environmental community's ability to improve models

This study was funded by the Natural Environment Research Council (NERC) Envision Doctoral Training Programme, project number 2929066.

collectively.

Many of these challenges stem from the inherent complexity of environmental ecosystems. Modelling these systems is highly interdisciplinary, requiring extensive collaboration across domains, and the field evolves rapidly as new scientific insights and datasets become available [8]. Environmental systems are inherently complex and can be understood and simulated using different conceptualisations, leading to the development of many alternative and isolated models [9]. Furthermore, environmental systems vary widely across space and context, which means a model suitable for one region or case may not be applicable elsewhere [10]. This spatial and contextual diversity results in a proliferation of specialised models [10]. Another important aspect is that most environmental modellers are not formally trained as software engineers [7], which can further complicate the adoption of robust development practices. Nevertheless, some of these are also common challenges when working with large, monolithic systems. Given these characteristics and challenges, an architectural approach that supports maintainability, scalability, and collaborative development is needed. Contemporary and flexible architectures, such as microservices architecture (MSA), offer a modular and scalable approach to software development that aligns well with these needs. MSA refers to designing a system as a collection of small, self-contained services [11]. Each service operates independently in its own process and communicates with others using lightweight communication protocols such as HTTP, typically exposed through APIs [11].

MSA offers several advantages that align well with the needs of environmental modelling [12], [13]. It supports loose coupling and modularity, allowing components to be modified or replaced with minimal impact on the wider system. It enables technology heterogeneity, allowing each service to use the most appropriate programming languages, frameworks, and other components of the software stack for its specific tasks. MSA also provides independent scalability, allowing only the most demanding components to be scaled, and supports independent development and deployment, which fits the interdisciplinary nature of environmental modelling and facilitates continuous improvement.

The objective of this study is to explore the feasibility of MSA in environmental modelling, using a case study to illustrate the approach in a practical setting. The study also examines the potential benefits of MSA in addressing current challenges in environmental modelling. Specifically, this study provides: (1) an initial demonstration of how MSA principles can be applied to environmental modelling, and (2) a case-study implementation that shows the potential of MSA through observed improvements in maintainability, scalability, and collaborative development. This work serves as a starting point for further research into transitioning environmental models from monolithic architectures to microservice-based solutions.

The rest of the paper is organised as follows: We first review existing efforts in the environmental community to address the challenges identified above. Next, in the Case Study section, we introduce a monolithic environmental model as an example and explain both the original monolithic design and the new microservice-based structure. In the Evaluation section, we assess the feasibility of applying a MSA to this model from both qualitative and quantitative perspectives. The qualitative evaluation focuses on how the MSA can improve the maintainability of environmental models and support for collaborative development. The quantitative evaluation examines model performance, including response time and throughput, as well as scalability and resource utilisation, specifically CPU and memory usage. In the Discussion and Future Work section, we reflect on the limitations of this work, discuss the challenges of applying a MSA to environmental models, and suggest directions for future work.

II. Existing Efforts in the Environmental Modelling Community

For decades, environmental modellers have worked to address the challenges outlined in the Introduction section by developing modular, distributed, and interoperable modelling systems [14]. Much of the efforts have focused on modularity to improve maintainability and reusability which enables models and components to be shared and adapted, and updated as scientific knowledge advances.

These efforts have progressed from early object-oriented programming techniques to more recent component-based environmental modelling frameworks (EMFs). Examples of EMFs include [15]: Modular Modelling System (MMS), Dynamic Integration Architecture System (DIAS), Interactive Component Modelling System (ICMS), Tarsier, Spatial Modelling Environment (SME), Object Modelling System (OMS), European Open Modelling Interface and Environment (OpenMI) initiative, and Community Surface Dynamics Modelling System (CSDMS). In EMFs, models are structured as reusable components by interface standards in a given framework. Once developed within a specific EMF, these components can be reused across different models that adhere to the same framework, requiring minimal adaptation [16].

However, a significant challenge is the issue of framework invasiveness. Model code often becomes dependent on a particular framework, which complicates migration to other frameworks and hinders the reuse of models between different groups or organizations, especially when refactoring is necessary [17]. This dependence requires users to learn or adapt to specific frameworks, making collaboration and model sharing more complicated.

The concept of loosely coupled, distributed models has also existed in environmental modelling for some time. One of the most influential early formalisations was the Model Web [18], which was initially proposed for ecological forecasting to improve the interoperability of ecological and related models. They introduced a 5–10 year vision, describing an ideal implementation of the Model Web, which included key components such as a distributed network of interoperable models, datasets, and sensors; the integration of physical, chemical, biological, and ecological processes; and models

and datasets that are maintained, operated, and served independently [18]. Nativi et al. [19] expands on the Model Web concept by proposing specific ideas for its development. It argues that Service-Oriented Architecture (SOA) is a suitable technological framework for implementing the Model Web, as SOA abstracts internal details, facilitates communication via open standards, and enhances interoperability through non-proprietary technologies. This flexibility makes it easier to integrate new models. By leveraging SOA, the Model Web can expose models as services, transitioning from monolithic modelling software to a Model as a Service (MaaS) paradigm. Although this concept is theoretically promising, it largely remains at a conceptual stage.

Later, there have been significant efforts to use SOA for model integration. In these SOA-based solutions, web services are employed for communication and coordination between models. A common strategy involves the use of model wrappers, which refer to interfaces that encapsulate individual models and expose them as web services. For example, Belete et al. [20] proposed the Distributed Model Integration Framework (DMIF) to link multidisciplinary and heterogeneous models in a meaningful way by wrapping them as web services. Similarly, the Experimental modelling Environment for Linking and Interoperability (EMELI) offers a platform to integrate BMI-enabled web service models [21]. However, services in these frameworks are often implemented at a coarse granularity, encompassing multiple functionalities within a single, large service. This approach reduces flexibility, as modifying a single function may require updating the entire service. It is also common for multiple services to share a single underlying database, resulting in tight coupling and hindering scalability. Moreover, these SOA-based solutions have primarily focused on enabling initial model integration, and less attention has been given to long-term maintainability, scalability, and fine-grained customization. There is also a need to transition from proprietary or highly specialised solutions to open and universal approaches to make models accessible to the wider community. Overall, these limitations in existing solutions motivate the exploration of more flexible architectural approaches.

III. Case Study

Our case study presents a bioturbation model, designed to simulate the vertical dispersion of a contaminant within a soil profile as a result of earthworm bioturbation. The model has previously been applied to predict the bioturbation of Ag_2S nanoparticles [22]. Understanding how earthworms affect the movement of different contaminants is essential for predicting the fate and exposure of these contaminants in the environment. The purpose of this model is to quantitatively describe the redistribution of a contaminant within a stratified soil profile, illustrating how concentrations change across different soil layers over time.

The original model [23] follows a modular monolithic architecture (Fig.1A), consisting of two main modules: soil and model. It is implemented in Python and is distributed as an installable Python package. The soil layer class represents a single layer within the soil profile. Each instance of it is initialised with specific parameters needed for subsequent computing. The soil profile class constructs the soil profile by instantiating the specified number of soil layer objects with the provided parameters. Inside the soil profile class, there is a key method for simulating the process of bioturbation over a time interval, and updating the contaminant concentrations in soil layers accordingly. The model module uses the soil module to perform the simulation calculations and manages the overall workflow, including running the simulation, storing the results in a database, and using the stored data to visualise the contaminant distribution across soil layers over time through its built-in results plotting function. (Fig.1A).

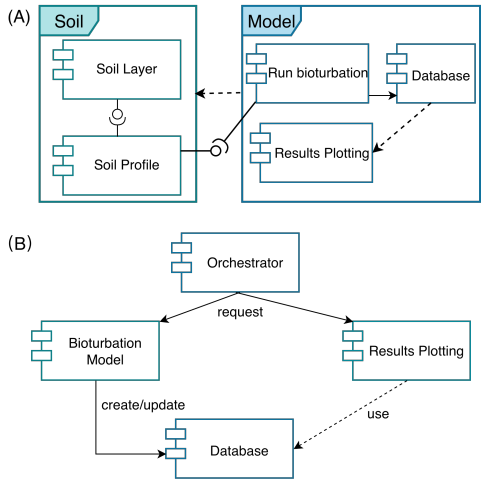

Fig. 1. Component diagram of original (A) and microservice-based (B) bioturbation model

In the microservice-based model (Fig.1B), the functionality of the original model is separated into two microservices: bioturbation model and result plotting. The bioturbation model microservice takes user input of the parameters needed for the simulation, initialises soil data in a database, performs the simulation, and stores the results into the database. The result plotting microservice retrieves the stored results and generates the same visualisation as the original model. An orchestrator is used to manage the end-to-end simulation workflow by interacting with the two microservices for model computation and data visualisation. Both microservices are implemented in Python, using the Flask framework. The microservices communicate via RESTful HTTP calls. The databases in both original and microservice-based models are implemented using MongoDB.

IV. Feasibility Assessment

In this section, we first discuss how the MSA can qualitatively enhance the maintainability of environmental models, as well as better support collaboration during their development, with reference to our specific use case. Then, we demonstrate that the MSA can quantitatively enhance scalability

and improve the model's performance metrics under certain conditions, while also being more resource-efficient.

A. Qualitative Evaluation

1) Maintainability: In the MSA, clear boundaries are established between functionalities, ensuring that updates to one service do not risk impacting others. For instance, when adding a new plot type or modifying the visualisation style, only the plotting microservice needs to be updated and redeployed, rather than the entire application. In contrast, the original monolithic model intertwines result visualisation logic with the simulation workflow, which complicates maintenance. With MSA, each service contains only the code relevant to its specific responsibility, enabling quick identification and comprehension of logic, and thereby simplifying code updates and maintenance. Furthermore, dependency management is also streamlined, as dependencies can be upgraded or changed within one service without introducing conflicts in others. Overall, compared to a modular monolithic architecture, the MSA provides superior maintainability by isolating concerns and supporting the independent evolution of individual components.

2) Benefits to collaborative development: The MSA also brings significant collaboration benefits. By separating the application into distinct services, cross-team dependencies can be reduced, as contributors only need to work with the microservice relevant to their tasks, rather than the entire codebase. This approach also makes it possible for teams to choose different programming languages or technologies for each service, if desired. For users who wish to adapt the model to their own needs, the modular design helps them identify and modify only the necessary components. Additionally, integrating the model into a larger system is more straightforward, since each microservice has a clear and well-defined interface. Overall, the MSA supports parallel development and makes it easier to extend or reuse the model in different contexts.

B. Quantitative Evaluation

1) Testing methods: The performance evaluation consisted of two stages: (a) Local baseline testing and (b) Cloud deployment testing.

a) Local baseline testing: We first compared the monolithic and microservice implementations of the bioturbation model in a controlled local environment. To ensure consistency, both architectures were executed using Docker, a widely used containerisation platform that packages applications and their dependencies into isolated, portable containers for reliable execution across different environments [24]. The monolithic model was deployed as a single container, while the microservice-based model consisted of two containers (one for the model service and one for the plotting service, with one instance of each). The tests were executed on a machine with the following specifications:

- Operating System: macOS Sequoia 15.3.1
- Processor (CPU): Apple M2 Pro, 10-core
- Memory (RAM): 16 GB
- Java Version: 21.0.6 (LTS)

Load generation was performed using Apache JMeter (version 5.6.3), configured with 10 concurrent threads issuing 100 requests each (i.e., a total of 1,000 requests). We measured error rate, number of successful requests over time, and response time over time using JMeter. Additionally, CPU and memory usage were monitored using Docker stats.

b) Cloud deployment testing: In the second stage, we evaluated how each architecture scales under a cloud deployment on AWS Fargate. The monolithic model was deployed as a single task with 1 vCPU and 2 GB of RAM. The microservice-based model was deployed as two services: a model service (1 task, 0.25 vCPU, 0.5 GB RAM) and a plotting service (3 tasks, each 0.25 vCPU, 0.5 GB RAM). For both architectures, autoscaling added instances when average CPU utilisation exceeded 70%. The monolith and the model service scaled between 1–4 tasks, and the plotting service scaled between 3–12 tasks. This configuration ensured resource parity at both bounds: at minimum scale each architecture provisioned 1 vCPU / 2 GB, and at maximum scale each provisioned 4 vCPU / 8 GB.

We stressed each deployment using JMeter at 10, 100, and 500 concurrent threads, with ramp-up periods of 5 s, 50 s, and 250 s respectively, and then sustained the load for 10 minutes. We recorded response time, throughput, error rate using JMeter, and the number of active task instances over time from CloudWatch.

To simulate soil data threads, dummy data is used for performance testing (Table I). These data sets vary in the number of soil layers, soil layer depths, earthworm density, and the bioturbation fitting parameter (beta). They all represent parameters that are likely to be encountered in real world situations. The test use case involves users providing these data as input and generating a plotted visualisation of contaminant concentration changes across soil layers over time.

TABLE I
TESTING DATA USED FOR THE PROJECT

n_ soil_layers	soil_ layer_depth	initial_conc	earthworm density	beta
3	0.10	[5e-9, 3e-9, 1e-9]	[5, 5, 5]	1.0e-09
4	0.10	[4e-9, 0, 0, 0]	[20, 20, 20, 20]	1.0e-08
4	0.10	[4e-9, 3e-9, 2e-9, 1e-9]	[30, 25, 20, 15]	3.0e-08
5	0.15	[3e-9, 0, 0, 0, 0]	[5, 10, 15, 10, 5]	2.0e-08
6	0.20	[5e-9, 1e-9, 0, 0, 0, 0]	[15, 15, 20, 25, 20, 15]	5.0e-08

Note: initial_conc and earthworm_density are vector values for each layer.

2) Results: This section presents and discusses the results obtained from both the local baseline testing and the cloud deployment testing.

a) Local baseline testing: Overall, the monolithic model exhibited higher resource consumption compared with the

microservice-based model. The monolithic model demonstrated greater CPU utilisation (mean: 170.25%) than the microservice-based model (mean: 144.68%) (Fig.2A) and, for most of the execution period, consumed more memory (mean: 966.28 MB) than the microservice-based one (mean: 810.14 MB) (Fig.2B). Within the MSA, the plotting service was the primary contributor to overall resource usage, whereas the model service showed negligible CPU and memory utilisation (Fig.2). This observation informed the configuration of the second testing stage, in which additional resources were initially allocated to the plotting service to better reflect its higher computational demand.

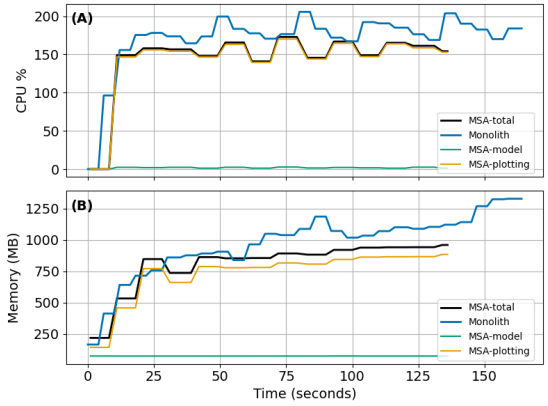

Fig. 2. Comparison of CPU (A) and memory (B) utilisation for the monolithic and microservice-based bioturbation model

The error rate for both architectures was 0%, showing stable execution. In terms of performance, the microservice-based model achieved lower average response times (mean: 1358 ms) compared with the monolithic model (mean: 1642 ms) (Fig.3A). Correspondingly, the microservice-based model sustained a higher throughput (average: 7.3 requests per second) than the monolithic model (6.0 requests per second) (Fig.3B). This performance difference can be attributed to the fact that both microservice containers were executed on the same host machine, where inter-service communication occurs via Docker's internal network, incurring minimal latency. Furthermore, by separating the model and plotting components into distinct services, the system enables concurrent execution, whereas the monolithic configuration processes these tasks sequentially.

b) Cloud deployment testing: Across all tests, the error rate was 0%, confirming that both architectures handled the applied workloads reliably. As the load increased, the autoscaling policies were triggered, and the corresponding total provisioned vCPU and RAM are shown in Fig.4. At the lowest load (10 threads), the monolithic model scaled from one to two instances, while the microservice-based model remained at its initial configuration (1v CPU, 2 GB RAM) (Fig.4). As the workload increased, the monolithic model scaled as a single unit, whereas in the microservices deployment, the model service consistently maintained a single instance, and the plotting service scaled from three instances under low load

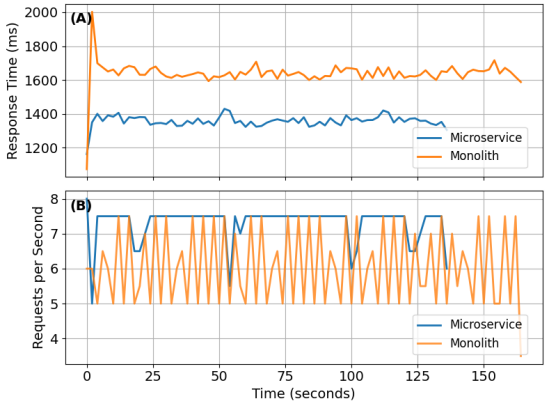

Fig. 3. Comparison of response time (A) and completed request per seconds (B) for the monolithic and microservice-based bioturbation model over time

to a maximum of five under heavier workloads. Consequently, the total allocated capacity for the microservice-based model (1.5 vCPU, 3 GB RAM) was lower than that of the monolithic model (2 vCPU, 4 GB RAM) (Fig.4). The ability to scale services independently demonstrates the resource efficiency and scalability of the MSA.

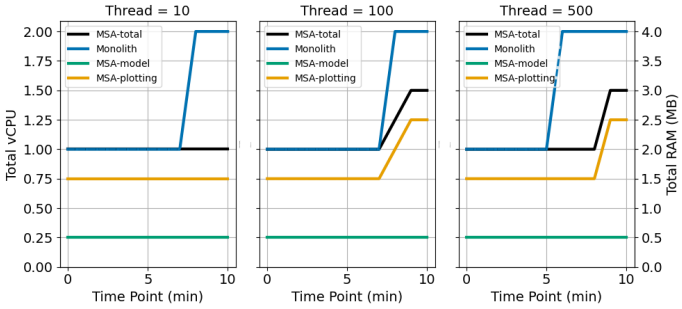

Fig. 4. CPU (vCPU) and memory (RAM) utilisation of the monolithic and microservice-based bioturbation model under different workloads

We also compared the throughput and response time across different load levels (Fig.5). At 10 threads, both architectures exhibited comparable performance; however, at this stage, while the monolithic model had already begun scaling up (Fig.4), the microservice-based model achieved similar throughput with lower resource consumption. When the load increased to 100 threads, the microservice-based model began to scale, resulting in a higher throughput (4.2 requests per second) compared with the monolithic model (3.1 requests per second) (Fig.5A). At 500 threads, the performance of both architectures decreased, with the microservice-based model having 3.0 requests per second and the monolithic model 2.9 requests per second (Fig.5A).

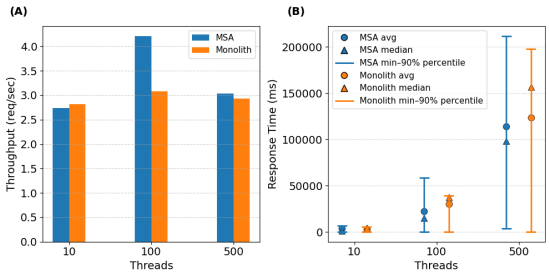

Fig. 5. Comparison of throughput (A) and response time (B) for the monolithic and microservice-based bioturbation model under different workloads

V. Discussion and Future Work

There have been several studies comparing the performance of systems with MSA and monolithic architecture. Jatkiewicz and Okrój [25] conducted a performance benchmark between microservice and monolithic applications deployed both locally and on Microsoft Azure. Their results showed that monolithic architectures generally outperformed microservices in terms of response time and throughput when both were executed on a single local or small cloud instance. Additionally, Berry et al. [26] conducted the migration of a monolithic web application and compared the performance between the two architectures. They found that replicating microservices improved performance, but even without scaling, the MSA outperformed the monolith in their study [26]. Similarly, in our study, the microservice-based model demonstrated better performance than the monolithic model in the local environment. When deployed in the cloud with equivalent resources, the microservice-based model also outperformed the monolithic model. Jatkiewicz and Okrój [25] also acknowledged that MSA benefits significantly from horizontal scaling. In scenarios where services were distributed across multiple cloud instances, performance improved substantially, and exceeded the monolithic system in some cases [25]. Our results confirms this observation: when auto-scaling was triggered in the microservice-based model, we observed a greater performance advantage over the monolithic model. In our study, under low load conditions, the monolithic model performed slightly better than the microservice-based model, which aligns with findings from other studies indicating that monolithic applications tend to perform better under light workloads, whereas MSA exhibit superior performance under heavy load conditions [27], [28]. In terms of resource consumption, consistent with our observations, previous studies have found that MSA achieve more efficient scaling by allowing only the specific service that needs to handle increased user requests to scale, whereas in monolithic architectures, the entire application must be scaled [27], [29].

However, performance enhancement is not necessarily the primary motivation for adopting a MSA in environmental modelling. While such distributed architectures can provide computational benefits in some cases, their advantages also lie in better maintainability, ease of collaboration, and greater flexibility. MSA promotes platform and hardware independence, allowing services to be deployed across diverse environments. It also enables developers to build individual components using different software stacks, selecting the most suitable technologies for each task. Furthermore, it supports the flexible extension and scalable addition of new features and functions as modelling workflows evolve. Although we have briefly rationalised some of these qualitative benefits in the Qualitative Evaluation section, they still require further empirical validation, potentially through user studies and systematic evaluation frameworks. Metrics such as development effort, ease of integration, and fault isolation should be explored to quantify the practical impact of microservices in environmental modelling workflows.

The present study used a small-scale implementation as a proof of concept for the feasibility of applying a MSA to environmental modelling. While this demonstration provides initial insights, it is insufficient to comprehensively capture the full potential and challenges associated with microservices. Future research should focus on applying microservices to larger and more complex environmental models to better assess their scalability, maintainability, and performance in real-world applications. Furthermore, the implementation of MSA involves a diverse range of technology stacks and design patterns. The current study provides a basic implementation, but future research should also investigate alternative technology choices.

Despite the theoretical advantages, transitioning from monolithic architectures to microservices can present significant challenges. A key issue is the effective decomposition of existing monolithic model codebases into microservices [30]. Determining the optimal service granularity is particularly complex, as excessively fine-grained services can introduce unnecessary inter-service communication overhead, whereas coarse-grained services may negate the benefits of modularity. This trade-off requires further research on domain-specific decomposition strategies and best practices for microservice design.

Beyond technical issues, there are also domain-specific challenges when applying a MSA in the context of environmental modelling. For many in the environmental science community, adopting a MSA may not be straightforward. This challenge can be viewed from two perspectives: greenfield development and brownfield development [31].

In greenfield scenarios, where new models are being developed, the barrier is often more about existing habits and the learning curve. Many environmental scientists are used to working in a certain way, and learning how to build systems using a MSA may require time and training. Importantly, environmental modellers should not be expected to work as professional software engineers. This, however, requires a paradigm shift in modelling practices and coordinated efforts to develop frameworks that are fit for environmental science needs, tailored to user requirements, and provide an inclusive and accessible user experience. In recent years, several efforts have been made to lower the technical barriers to adopting microservices, making the architecture more accessible. For

example, a recent work integrates the MSA with the low-code/no-code paradigm, and proposed a platform that allows non-developers to easily build and deploy applications [32]. Artificial intelligence is also beginning to play a growing role in lowering the barriers to microservice development [33]. One study, for example, demonstrates how large language models (LLMs) can assist in microservice development. They present a system that uses an LLM to generate a RESTful microservice from an API description [34]. There are also microservices generation tools such as JHipster, MicroBuilder, and MAGMA [35]–[38]. However, despite these efforts, adoption of these tools remains limited communities without a software engineering background, and learning to use them is still non-trivial for users without such a background. Complementing these tool-based approaches, the emergence of Research Software Engineers (RSEs) offers a people-centred bridge for this skills gap [39]. RSEs combine software-engineering expertise with domain knowledge [39]. In the context of environmental modelling, they can co-develop models, design and document APIs, establish containerisation and continuous-integration/continuous-deployment (CI/CD) pipelines, support deployment, and provide training and scaffolding/templates that reduce the learning curve.

In brownfield scenarios, where legacy models already exist, the challenges faced in greenfield contexts apply alongside additional challenges. The cost and effort required to analyse and refactor the existing codebase can be substantial. In some cases, the code may not be well documented (or not documented at all), or not follow software engineering best practices, making it questionable whether migration to the MSA is worth the investment. To support informed decision-making in these cases, several frameworks and guidelines have been proposed to help organizations evaluate whether migrating from a monolithic to a MSA is feasible and worthwhile [40]–[42]. However, practical, domain-specific decision criteria tailored to environmental modelling are still lacking.

While this study is framed within environmental modelling, the potential applicability of microservices extends to traditional scientific modelling more broadly. Many scientific modelling communities share similar software characteristics and needs: models are implemented as monolithic system that needs to evolve incrementally to incorporate new domain knowledge or emerging research demands [43]. Over time, such systems increasingly difficult to maintain or scale. As Johanson and Hasselbring [44] noted, scientific software is often developed within isolated research cultures that prioritise performance and scientific correctness, while systematic adoption of modern software engineering practices remains limited due to cultural expectations, tool constraints, and the perceived overhead of restructuring legacy code. These shared conditions suggest that the challenges and trade-offs discussed here are also faced across scientific modelling communities. Future work will investigate additional use cases and refactoring strategies for transitioning monolithic scientific models to MSAs, with the aim of enabling next-generation environmental models that are more maintainable, scalable, and better support collaborative development.

VI. Conclusion

Environmental models have traditionally relied on monolithic architectures, which face challenges in achieving scalability, maintainability, and collaborative development. In this context, we suggest that MSA offer advantages that can help address these limitations. Our study demonstrates the feasibility of applying MSA in environmental modelling through a case study of a small-sized model. The model refactored to the MSA exhibited overall improved performance, and a significant advantage in scalability and resource efficiency. Although still in an exploratory stage, these findings suggest the promising potential of the MSA in environmental modelling and provide a starting point for further exploration and extension in future generations of environmental models.

References

[1] D. P. Holzworth, N. I. Huth, and P. G. deVoil, "Simple software processes and tests improve the reliability and usefulness of a model," *Environmental Modelling & Software*, vol. 26, no. 4, pp. 510–516, 2011.

[2] O. David, J. Ascough, W. Lloyd, T. Green, K. Rojas, G. Leavesley, and L. Ahuja, "A software engineering perspective on environmental modeling framework design: The object modeling system," *Environmental Modelling & Software*, vol. 39, pp. 201–213, 2013, thematic Issue on the Future of Integrated Modeling Science and Technology.

[3] R. Schwemmle, H. Leistert, A. Steinbrich, and M. Weiler, "Roger v3.0.5 - a process-based hydrological toolbox model in python," *Geoscientific Model Development*, vol. 17, no. 13, pp. 5249–5262, 2024.

[4] D. P. Holzworth, N. I. Huth, and P. G. de Voil, "Simplifying environmental model reuse," *Environmental Modelling & Software*, vol. 25, no. 2, pp. 269–275, 2010.

[5] K. de Jong, D. Panja, M. van Kreveld, and D. Karssenberg, "An environmental modelling framework based on asynchronous many-tasks: Scalability and usability," *Environmental Modelling & Software*, vol. 139, p. 104998, 2021.

[6] M. E. Emetere, M. Akinyemi, T. Oluwafemi, T. Akinlusi, M. Allen, and S. Adewole, "Role of software engineering processes to develop environmental model," in *Proceedings of the World Congress on Engineering and Computer Science*, vol. 1, 2016.

[7] K. Gregor, B. F. Meyer, T. Gaida, V. Justo Vasquez, K. Bett-Williams, M. Forrest, J. P. Darela-Filho, S. Rabin, M. Longo, J. R. Melton, J. Nord, P. Anthoni, V. Bastrikov, T. Colligan, C. Delire, M. C. Dietze, G. Hurtt, A. Ito, L. T. Keetz, J. Knauer, J. Köster, T.-S. Lin, L. Ma, M. Minvielle, S. Olin, S. Ostberg, H. Shi, R. Schnur, U. Schönenberger, Q. Sun, P. E. Thornton, and A. Rammig, "Best practices in software development for robust and reproducible geoscientific models based on insights from the global carbon project models," *EGUsphere*, vol. 2025, pp. 1–44, 2025.

[8] G. F. Laniak, G. Olchin, J. Goodall, A. Voinov, M. Hill, P. Glynn, G. Whelan, G. Geller, N. Quinn, M. Blind, S. Peckham, S. Reaney, N. Gaber, R. Kennedy, and A. Hughes, "Integrated environmental modeling: A vision and roadmap for the future," *Environmental Modelling & Software*, vol. 39, pp. 3–23, 2013, thematic Issue on the Future of Integrated Modeling Science and Technology.

[9] M. P. Clark, B. Nijssen, J. D. Lundquist, D. Kavetski, D. E. Rupp, R. A. Woods, J. E. Freer, E. D. Gutmann, A. W. Wood, L. D. Brekke, J. R. Arnold, D. J. Gochis, and R. M. Rasmussen, "A unified approach for process-based hydrologic modeling: 1. modeling concept," *Water Resources Research*, vol. 51, no. 4, pp. 2498–2514, 2015.

[10] G. S. Blair, K. Beven, R. Lamb, R. Bassett, K. Cauwenberghs, B. Hankin, G. Dean, N. Hunter, L. Edwards, V. Nundloll, F. Samreen, W. Simm, and R. Towe, "Models of everywhere revisited: A technological perspective," *Environmental Modelling & Software*, vol. 122, p. 104521, 2019.

[11] J. Lewis and M. Fowler, "Microservices," Available at: https://martinfowler.com/articles/microservices.html (Accessed: 26 January 2025), no date, martinfowler.com. [Online]. Available: https://martinfowler.com/articles/microservices.html

[12] P. D. Francesco, I. Malavolta, and P. Lago, "Research on architecting microservices: Trends, focus, and potential for industrial adoption," in *2017 IEEE International Conference on Software Architecture (ICSA)*, 2017, pp. 21–30.

[13] P. Di Francesco, P. Lago, and I. Malavolta, "Architecting with microservices: A systematic mapping study," *Journal of Systems and Software*, vol. 150, pp. 77–97, 2019.

[14] G. E. Tucker, E. W. H. Hutton, M. D. Piper, B. Campforts, T. Gan, K. R. Barnhart, A. J. Kettner, I. Overeem, S. D. Peckham, L. McCready, and J. Syvitski, "Csdms: a community platform for numerical modeling of earth surface processes," *Geoscientific Model Development*, vol. 15, no. 4, pp. 1413–1439, 2022.

[15] R. M. Argent, A. Voinov, T. Maxwell, S. M. Cuddy, J. M. Rahman, S. Seaton, R. Vertessy, and R. D. Braddock, "Comparing modelling frameworks–a workshop approach," *Environmental Modelling & Software*, vol. 21, no. 7, pp. 895–910, 2006.

[16] W. Lloyd, O. David, J. C. Ascough II, K. W. Rojas, J. R. Carlson, G. H. Leavesley, P. Krause, T. R. Green, and L. R. Ahuja, "Environmental modeling framework invasiveness: Analysis and implications," *Environmental Modelling & Software*, vol. 26, no. 10, pp. 1240–1250, 2011.

[17] ——, "An exploratory investigation on the invasiveness of environmental modeling frameworks," *School of Engineering and Technology Publications*, vol. 18, 2009.

[18] G. N. Geller and W. Turner, "The model web: a concept for ecological forecasting," in *2007 IEEE International Geoscience and Remote Sensing Symposium*, 2007, pp. 2469–2472.

[19] S. Nativi, P. Mazzetti, and G. N. Geller, "Environmental model access and interoperability: The geo model web initiative," *Environmental Modelling & Software*, vol. 39, pp. 214–228, 2013.

[20] G. F. Belete, A. Voinov, and J. Morales, "Designing the distributed model integration framework – dmif," *Environmental Modelling & Software*, vol. 94, pp. 112–126, 2017.

[21] P. Jiang, M. Elag, P. Kumar, S. D. Peckham, L. Marini, and L. Rui, "A service-oriented architecture for coupling web service models using the basic model interface (bmi)," *Environmental Modelling & Software*, vol. 92, pp. 107–118, 2017.

[22] M. Baccaro, S. Harrison, H. van den Berg, L. Sloot, D. Hermans, G. Cornelis, C. A. van Gestel, and N. W. van den Brink, "Bioturbation of ag2s-nps in soil columns by earthworms," *Environmental Pollution*, vol. 252, pp. 155–162, 2019.

[23] S. Harrison, V. D. Keller, R. J. Williams, M. Hutchins, and S. Lofts, "Bioturbation model (1.0.1)," 2022. [Online]. Available: https://doi.org/10.5281/zenodo.6387842

[24] Docker Inc., "Docker overview," https://docs.docker.com/get-started/docker-overview/, 2025, accessed: 2025-11-21.

[25] P. Jatkiewicz and S. Okrój, "Differences in performance, scalability, and cost of using microservice and monolithic architecture," in *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, ser. SAC '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 1038–1041.

[26] V. Berry, A. Castelltort, B. Lange, J. Teriihoania, C. Tibermacine, and C. Trubiani, "Is it worth migrating a monolith to microservices? an experience report on performance, availability and energy usage," in *2024 IEEE International Conference on Web Services (ICWS)*, 2024, pp. 944–954.

[27] P. Mangwani, N. Mangwani, and S. Motwani, "Evaluation of a multi-tenant saas using monolithic and microservice architectures," *SN Computer Science*, vol. 4, no. 2, p. 185, 2023.

[28] K. Gos and W. Zabierowski, "The comparison of microservice and monolithic architecture," in *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*. IEEE, 2020, pp. 150–153.

[29] K. Adrio, C. N. Tanzil, M. C. Lianto, and Z. E. Rasjid, "Comparative analysis of monolith, microservice api gateway and microservice federated gateway on web-based application using graphql api," in *2023 10th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, 2023, pp. 654–660.

[30] G. Blinowski, A. Ojdowska, and A. Przybyłek, "Monolithic vs. microservice architecture: A performance and scalability evaluation," *IEEE Access*, vol. 10, pp. 20 357–20 374, 2022.

[31] M. Garriga, "Towards a taxonomy of microservices architectures," in *Software Engineering and Formal Methods*, A. Cerone and M. Roveri, Eds. Cham: Springer International Publishing, 2018, pp. 203–218.

[32] M. AIT Said, A. Ezzati, and S. Arezki, "Microservices, a step from the low-code to the no-code," in *International Conference on Advanced Intelligent Systems and Informatics*. Springer, 2022, pp. 779–788.

[33] D. Narváez, N. Battaglia, A. Fernández, and G. Rossi, "Designing microservices using ai: A systematic literature review," *Software*, vol. 4, no. 1, 2025.

[34] S. Chauhan, Z. Rasheed, A. M. Sami, Z. Zhang, J. Rasku, K.-K. Kemell, and P. Abrahamsson, "Llm-generated microservice implementations from restful api definitions," *arXiv preprint arXiv:2502.09766*, 2025.

[35] JHipster. (2024) Doing microservices with jhipster. Accessed: 2025-06-22. [Online]. Available: https://www.jhipster.tech/microservices-architecture/

[36] (n.d.) Microbuilder. Accessed: 2025-06-22. [Online]. Available: https://thoughtworksinc.github.io/microbuilder/1-overview.html

[37] P. Wizenty, J. Sorgalla, F. Rademacher, and S. Sachweh, "Magma: Build management-based generation of microservice infrastructures," in *Proceedings of the 11th European conference on software architecture: companion proceedings*, 2017, pp. 61–65.

[38] H. Wei, N. H. Madhavji, and J. Steinbacher, "Magen: A tool for generating microservice-based skeleton applications," in *Proceedings of the 33rd Annual International Conference on Computer Science and Software Engineering*, 2023, pp. 4–13.

[39] I. A. Cosden, K. McHenry, and D. S. Katz, "Research software engineers: Career entry points and training gaps," *Computing in Science & Engineering*, vol. 24, no. 6, pp. 14–21, 2022.

[40] F. Auer, V. Lenarduzzi, M. Felderer, and D. Taibi, "From monolithic systems to microservices: An assessment framework," *Information and Software Technology*, vol. 137, p. 106600, 2021.

[41] P. I. Habib, A. S. Murdha, H. Agus, and Suhardi, "Architecture migration from monolithic to microservices: Developing readiness criteria," *IEEE Access*, vol. 12, pp. 194 630–194 645, 2024.

[42] H. M. Ayas, P. Leitner, and R. Hebig, "Facing the giant: a grounded theory study of decision-making in microservices migrations," ser. ESEM '21. New York, NY, USA: Association for Computing Machinery, 2021.

[43] E.-M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, and J. C. Carver, "Software engineering practices for scientific software development: A systematic mapping study," *Journal of Systems and Software*, vol. 172, p. 110848, 2021.

[44] A. Johanson and W. Hasselbring, "Software engineering for computational science: Past, present, future," *Computing in Science & Engineering*, vol. 20, no. 2, pp. 90–109, 2018.