OptiGob 2.0: A Model-Driven Approach for Exploring Agri-Sustainability Projections

Michael Mittermaier
michael.mittermaier@ur.de
Programming and
Software Engineering,
University of Regensburg
Regensburg, Germany

David Styles
david.styles@universityofgalway.ie
School of Biological
and Chemical Sciences,
University of Galway
Galway, Ireland

Judith Michael
judith.michael@ur.de
Programming and
Software Engineering,
University of Regensburg
Regensburg, Germany

ABSTRACT

Research software developed by domain experts enables scientific research by addressing domain-specific problems tailored to the domains and constraints of their respective fields. As such software evolves and its functional and quality requirements increase, applying software engineering practices becomes essential to manage the software system. An example of such research software is OptiGob, a decision-support tool to project CO_2e emissions among other properties for Ireland's agriculture, forestry, and other land use (AFOLU) sector. In this paper, we present the software engineering practices applied during the evolution of OptiGob to handle newly emerging requirements and a model-driven approach to establish a foundation for future extensions such as additional AFOLU systems and characteristics in the dataset and new application features. Our solution is based on MontiGem, a code generator for data-centric applications. This new version of OptiGob enables users to build and evaluate more complex AFOLU scenario configurations while managing the growing complexity of the software through model-driven engineering.

CCS CONCEPTS

• **Software and its engineering** → **Model-driven software engineering**; **Domain specific languages**; • **Theory of computation** → *Data modeling*; • **Social and professional topics** → **Socio-technical systems**.

KEYWORDS

Sustainability, Agriculture, Forestry, Simulation, Optimization, Research Software, MDE

ACM Reference Format:
Michael Mittermaier, David Styles, and Judith Michael. 2026. OptiGob 2.0: A Model-Driven Approach for Exploring Agri-Sustainability Projections. In *ACM/IEEE 29th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '26), Date,Location.* ACM, New York, NY, USA, 10 pages. https://doi.org/

MODELS Companion '26, Date, Location

https://doi.org/

1 INTRODUCTION

Across diverse scientific research domains, domain experts develop research software tailored to their specific demands and constraints to solve domain-specific problems. Advances in the field of large language models (LLMs) [22] assist them in the democratization of building software. This way, researchers can use their competencies in their respective fields to implement first prototypes. However, when requirements increase, the complexity of research software can become unmanageable with respect to extensibility, maintainability, and validation if software engineering practices are not applied [16].

An example for research software developed by domain experts is OptiGob [15], a decision support tool to assess AFOLU projections based on user-configured assumptions called scenarios. Soon after implementing the first prototype, domain experts discussed how to allow more detailed scenarios and enhance the evaluation options. This rise in complexity due to the newly introduced requirements elevates the need for software engineering principles to answer the research question:

RQ1 How can software engineering principles be applied to address evolving research software?

While the research software OptiGob started as a small research software project, over time, it has developed to a complex software system already with plans for the next extensions in terms of dataset and parameters, as well as additional features. This poses challenges to the maintainability and validity of the system over time. Thus, we aim to build a solid foundation for OptiGob's future development with model-driven engineering (MDE) methods to answer the research question:

RQ2 How can model-driven engineering support the sustainable evolution of research software under continuously changing domain requirements?

Our main contributions are:

(1) systematic refactoring of the OptiGob research software using established software engineering principles to address increasing functional complexity;
(2) a re-engineering process that derives an MDE-based implementation from the existing software
(3) insights into the benefits and challenges of applying MDE to improve the long-term maintainability and extensibility of evolving research software.

This paper is structured as follows. Section 2 provides an introduction to (i) the problem domain AFOLU and the new requirements for OptiGob, and (ii) the MDE techniques that we use in this work.

In Section 3, we describe the challenges for a new Optigob version, the applied principles of software engineering, and the resulting architecture. We follow with a description of the re-engineering process to transform OptiGob into a model-driven system in Section 4 and a specification of the developed system in Section 5. We discuss the main findings in Section 6, provide an overview of related work in Section 7, and conclude in Section 8.

2 BACKGROUND

To better understand the research software in focus, we first provide a rationale behind OptiGob in context of its application domain of AFOLU, and give an overview of MDE with the MontiGem code generator.

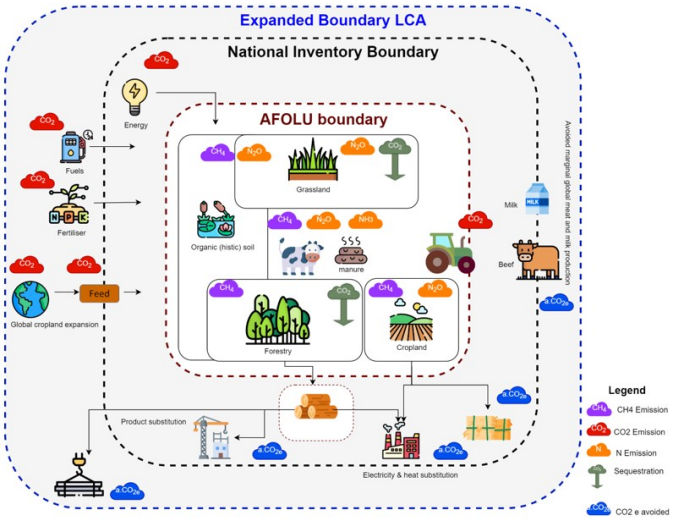

Figure 1: Overview of AFOLU systems and their output within the scope for OptiGob.

2.1 AFOLU and OptiGob

In climate sciences, AFOLU refers to the human use of and influence on land areas. Smith et al. identified this sector to be in a unique position due to its mitigation potential from (i) an enhancement of greenhouse gas removals, and (ii) a reduction of emissions through land and livestock management [32].

There are many projects centering around projecting parts of AFOLU. The food, agriculture, biodiversity, land-use and energy (FABLE) consortium [2] is a global network of researchers who develop national-scale food and land-use strategies to comply with global emissions and biodiversity goals. All-AFOLU models that operate at sufficient resolution to reflect national farming and land-use, land-use change, and forestry (LULUCF) systems and mitigation measures at a resolution useful for national policy making are sparse [23].

A pioneering model for assessing current land-use and evaluating land-use change scenarios was developed in New Zealand [7]. This model produces land-use maps but doesn't calculate greenhouse gas emissions itself where OptiGob does not map the land-use to specific locations, but does calculate emissions. Another related AFOLU scenario projecting model was presented for Brazil [17]. In contrast with mapping models or economically-driven models, OptiGob's goal is to enable users to explore wide range of biophysically feasible AFOLU pathways through time, in terms of sustainability performance. This can underpin wide ranging foresight analysis that is particularly relevant to support decision making in the context of a deeply uncertain future.

OptiGob 1.0 [15] was conceptually based upon research undertaken to explore specific land use scenarios for Ireland's policy-driven land use review process. Figure 1 provides a brief overview of all the systems included in these land use scenarios. The model enabled combinations of farming system types and configurations to be combined with other land uses, including forestry - and downstream value chain carbon storage in the case of forestry and bioenergy – and cattle activity to be scaled to a level compatible with predetermined climate targets in 2050 [34]. Subsequent engagement with policy stakeholders highlighted a need for more flexibility, to explore:

(i) wider combinations of land use options (e.g. the full scale of organic soil rewetting and afforestation rates possible), and;
(ii) multiple time way points representing sequential and sub-sectoral policy targets.

For example, national carbon budgets and EU LULUCF regulations [1] establish emission ceilings for agriculture [5] and LULUCF, respectively, in 2030. The EU Climate Law was revised in 2026 to include a target for a 90% reduction in greenhouse gas emissions across the bloc by 2040 [3]. Member states will put forward proposals to the European Commission on appropriate effort-sharing contributions to this target. In the case of Ireland, this requires evidence on greenhouse gas emissions from individual sectors out to 2040. Policymakers are also to understand the implications of possible separate targets and/or trading for short-lived climate pollutants such as methane and long-lived pollutants such as CO_2. Meanwhile, there is increasing attention on co-benefits and trade-offs between climate change mitigation and other objectives for land use, such as food production, reduced water and air pollution, increased biodiversity and resilience to climate and other shocks. Thus, there is a need to expand the flexibility of OptiGob to enable users to explore subsector activity levels based on multiple objectives over different time scales. OptiGob 2.0 needs to address this need.

The added requirements for a second version of OptiGob are:

(R1) *Maintainable and expandable AFOLU systems management.* Until now, the number of systems modeled in OptiGob was fixed. In the future, more AFOLU systems could be integrated into OptiGob, and the user should have the choice to limit the projections to selected systems to have a customizable scope in the scenario evaluation.

(R2) *Integrate dynamic configuration options to agricultural and organic soil systems.* Until now, the user has limited configuration options to decide one static timeline for individual AFOLU systems, this timeline was implicitly assumed to be linear changes between the baseline year 2020 and the target year 2050, and result in a new static equilibrium post 2050. Instead, the user should gain more control over constructing the scenario, and select way points with a user-defined

target year and system-specific properties (e.g., rewetting ratio for organic soils, and abatement type, and area, emission, or protein goals for agricultural systems). Similarly, the user should have more control over the timeline and scale of implementing the biomethane strategy.

(R3) *Complex scenario configurations should be storable.* Until now, configurations were simple enough to configure by hand at every run. With raising complexity of the scenarios, the user needs to be able to save and load complex configurations. This feature will also be useful for future requirements when optimization algorithms will be introduced to discover scenarios.

(R4) *Scenarios should be evaluated based on timelines.* Until now, the evaluation of configurations is limited to a comparison of the baseline year, and a target year. There is no data for the process in between. There is a need for a more detailed visualization of the projections to assess what actions need to be taken to achieve the goals set in the scenario and make assumptions on how realistic the scenario is.

(R5) *Introduction of more detailed AFOLU area modeling*. Until now, there is an area estimation for the individual systems in the scenario. There is, however, a need for understanding how areas affect each other in the timeline, and introduce additional AFOLU elements such as land use change emissions for cropland which was previously neglected.

2.2 MDE with MontiGem

Our aim in using a model-driven method in this approach is to better manage complexity through abstraction, improve software quality, and eliminate redundancies [33]. In particular, we use MontiGem, a generator framework specifically designed to build data-centric web applications [9].

Developing Software with MontiGem. MontiGem generates code from class diagrams and Graphical User Interface (GUI) models (see Figure 2). The software engineer provides (i) a class diagram to describe the domain-specific data structure, and (ii) GUI models in Montigem's domain-specific language (DSL) [19] to describe page layouts and handle the navigation within the frontend. Using these inputs, MontiGem generates a complex web application with a Spring Boot backend and an Angular frontend.

From here, the developer uses the TOP mechanism to add hand-written code for business logic, commands, controllers, etc. This mechanism means that MontiGem generates TOP classes for frontend and backend entities where hand-written code is already existing. The hand-written code can extend and override the TOP classes. This way, the software engineer not only reduces the volume of hand-written code by approximately 90%, they can also benefit from an intertwined frontend and backend. When an action from the frontend calls a service that modifies an object in the database (that MontiGem generates according to the domain class diagram), generated observers notify the frontend automatically to re-render the current Angular page. That way, the developer can focus on complex domain-specific logic instead of database management or communication between frontend and backend.

Retrofitting Software Systems with MontiGem. While most of the research surrounding MDE with MontiGem focuses on greenfield development, i.e., building a system from scratch, the case for this work, however, presents the problem of transferring an already running system into a model-driven software system. This process is called brownfield development [21]. Drave et al. proposed a methodology for retrofitting model-generated code into existing applications [14]. They suggest 16 activities in three phases, namely (i) *problem analysis and decomposition*, (ii) *domain-specific language engineering* and (iii) *application engineering and operation*, in a divide-and-conquer strategy. This breaks down the problem domain of an enterprise information system into smaller problems, and address those systematically with appropriate modeling languages and code generators. By using MontiGem, we already have a generator and several modeling languages in place to be reused. The current approach extends the work from [14] by discussing how GenAI additionally helps in this transformation.

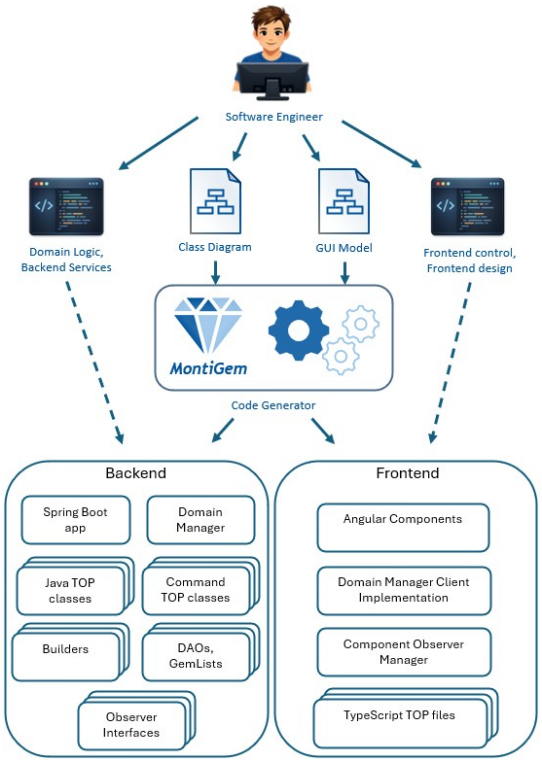

Figure 2: Overview of MontiGem-generated components (bottom) and the responsibilities of the software engineer (middle).

3 IMPROVING RESEARCH SOFTWARE WITH SOFTWARE ENGINEERING PRINCIPLES

Addressing *RQ1*, we used software engineering principles to improve the evolving research software OptiGob. Below, we describe the process by, first, addressing the underlying challenges caused by the additional requirements stated in Section 2.1, then, by naming the applied principles, followed by a description of the resulting architecture.

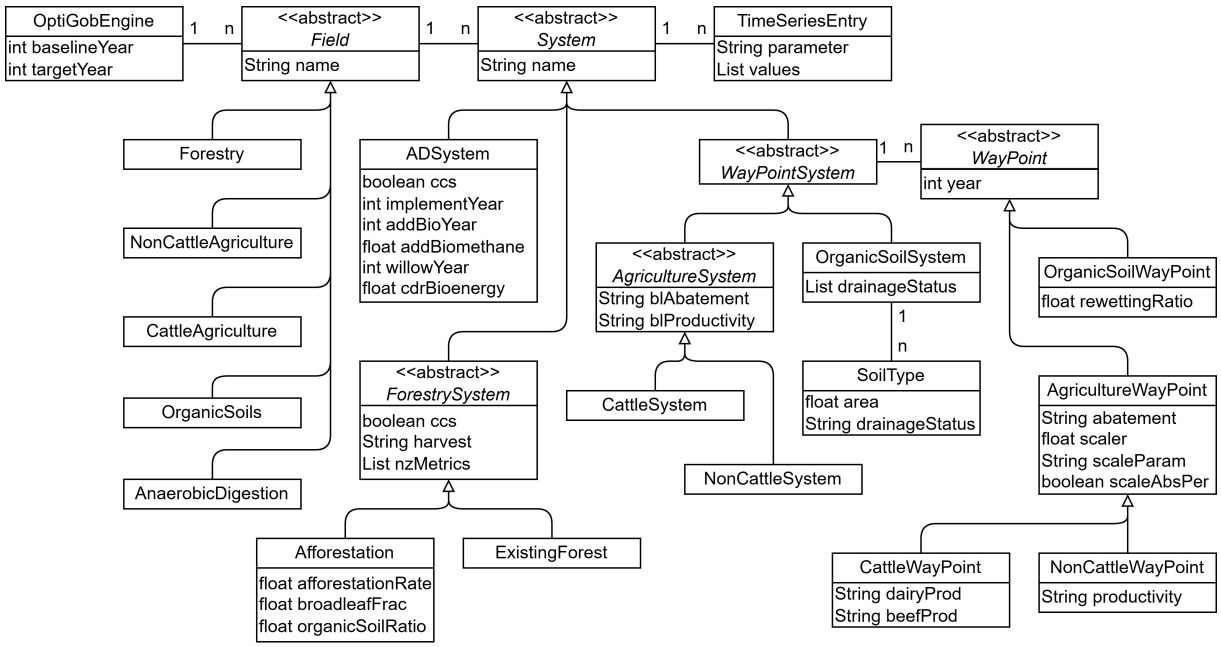

Figure 3: Visual representation of the data structure in OptiGob 2.0.

3.1 Challenges

The first version of OptiGob [15] already fulfills requirements such as setting up static AFOLU scenarios and automating CO_2e calculations. In terms of architecture, it uses an SQLite database to organize and store data produced with the land balance model GOBLIN [28], Streamlit [4] to build a user interface, and Pyomo [10] to calculate the number of cattle based on CO_2e budgets.

For the second version of OptiGob, we assessed the newly introduced requirements stated in Section 2.1 and formulated a technical challenge for each requirement to solve in OptiGob 2.0:

- **(C1)** Provide a maintainable and extensible data structure to prepare OptiGob for future changes in requirements to address *(R1)*
- **(C2)** Provide a dynamic and complex scenario configuration to maximize flexibility for a user-controlled scenario to address *(R2)*
- **(C3)** Provide gateways to the user through (i) the interface to enable download and upload of scenario configurations, and (ii) an API to allow the use of OptiGob through optimization algorithms to address *(R3)*
- **(C4)** Provide more details in the scenario evaluation through projecting timelines for each parameter in every AFOLU system to address *(R4)*
- **(C5)** Provide additional details within AFOLU projections by including area balancing to address *(R5)*

3.2 Applied Principles

The main focus was on allowing for more flexibility in OptiGob, both in designing an AFOLU scenario, but also in evaluating numerous parameters and goals over time while setting it up in a way to facilitate future expansions into OptiGob in terms of more AFOLU systems and/or parameters.

Upon analyzing the code base, it became clear that the code in the logic layer was not following software engineering practices when it comes to principles such as the *Open/Closed principle*, as the addition of new AFOLU systems required modifications to existing budget modules and database schema rather than isolated extensions. The *Single Responsibility principle* was also not followed, as the central data manager and orchestrator classes conflated data access, validation, and output formatting, as well as the *Don't Repeat Yourself principle*, as structurally similar scaler-lookup and aggregation logic was duplicated across independent sector modules. Followed by this analysis, we decided to re-structure the system entirely, so that we have a solid foundation to further extend on later on.

3.3 Resulting Architecture

While we retained the basic architecture of SQLite, Streamlit and Python for the domain logic, we identified the *abstract factory pattern* as the best option for modeling the application domain to address **(C1)**. We divided the different AFOLU areas into fields (i.e., forestry, cattle agriculture, non-cattle agriculture, organic soils, and anaerobic digestion (AD)) and systems within those fields (e.g., pigs, poultry, sheep, and crops for non-cattle agriculture). Figure 3 shows a class diagram of the application domain. With this design, we have substantial flexibility when it comes to the different constraints of projecting different systems, while maintaining a high degree of extensibility when more systems or even fields are introduced into OptiGob.

For organic soils and agricultural systems, we restructured the database from static timelines to individual way points to address

(C2). One way point consists of a number of conditions (e.g., abatement type and productivity for an agricultural way point) and a number of parameters such as greenhouse gas emissions, area, or produced protein. In the scenario configuration, the user can select not only the conditions for a specific target year, but also scale the data up or down by different parameters. This way, a user can configure multiple way points for each system individually, and the OptiGob engine connects those way points.

While transferring the database for those systems from static timelines to way points was feasible in the fields of agriculture and organic soils, the data for systems in forestry and AD remain static. For forestry, the required simulations for dynamic configurations would exceed computational restraints, and for AD, the AD strategy does per definition already have a specific timeline. The user still has some configuration options, e.g. setting the harvest rate, afforestation rate or setting a target year when to implementing the AD strategy.

With respect to **(C3)**, OptiGob 2.0 configurations are stored in JSON format. This enables extensibility, as well as an interface for the OptiGob engine from both the graphical user interface and optimization algorithms.

To address **(C4)**, OptiGob version 2.0 creates timelines for each parameter in every system. That way, the user may evaluate not only the difference for each parameter between the baseline and the target year, but also how the scenario got to that point.

Another newly introduced feature in OptiGob 2.0 to address **(C5)** concerns area balancing. In a nutshell, when systems are scaled up and they need more area, the engine ensures that this additional area is taken from another system so that the overall area does not change. The engine follows a set of rules and re-adjusts projections where needed. We explain this rule set at the end of Section 5.2. The change in the architecture to a timeline-based evaluation enables the flexibility to include area balancing into the projections and evaluations.

4 REVERSE ENGINEERING HAND-WRITTEN SOFTWARE WITH MDE

At this stage, the research software OptiGob 2.0 has already grown to a considerable size. As OptiGob will have to evolve further to include more entity classes, the complexity of adapting frontend, and handling a growing database schema will also increase. This motivates the use of model-driven techniques to decrease complexity through abstraction and therefore guarantee maintainability and extensibility in the future. In this project, we chose to use the code generator MontiGem in the re-engineering process.

In the following, we describe this re-engineering process, i.e., how we moved from a hand-written OptiGob implementation to a model-driven approach, to answer *RQ2*. Afterwards, we discuss the maintainability and extensibility of this model-driven approach.

4.1 Re-Engineering Process

Since we already had the hand-written implementation of OptiGob 2.0 as a description of the data structure, requirements, and design, we were able to keep the analysis stage of the re-engineering process rather short, and focus on these three steps: (i) transforming the data structure for the model-driven approach, (ii) model the GUI, and (iii) add the domain logic to perform AFOLU projections. In all three of these steps, we included Anthropic's Claude model Sonnet 4.6 [6] into the process for various tasks. In the discussion section, we provide insights on the use of LLMs in model-driven re-engineering processes.

Data Structure Transformation. The hand-written OptiGob version introduced in Section 3 was the start point of our re-engineering process. Here, we extracted a class diagram on paper from the data structure, and used Claude to transform that into CD4A [20, 27], MontiGem's DSL for class diagrams. Afterwards, it took several adaptations from there: For the timelines in each AFOLU system, we introduced the TimeSeriesEntry class instead of a Map object since MontiGem does not support them, yet. MontiGem is still under development and new functionality is constantly incorporated. Second, we added a configuration and evaluation class to facilitate the connection between frontend and backend.

GUI Modelling. For the frontend, we provided Claude with the Streamlit implementations of the hand-written version, and the documentation of MontiGem's GUI DSL [18, 19]. Based on this input and the guidance of the software engineer, Claude generated GUI models for each page step by step as well as a CSS file to imitate the appearance of the hand-written user interface.

Domain Logic. While MontiGem facilitates and accelerates the engineering process by generating a majority of the system based on GUI models and class diagrams, there are still backend services, that are domain-specific, and cannot be automatically generated. In case of OptiGob, the most crucial backend service concerns the calculation of AFOLU predictions. Here, Claude was able under proper guidance to translate these calculations step by step from the hand-written Python version into a Java backend service.

4.2 Maintainability and Extensibility with MDE

MontiGem generates a full-stack reactive web application with all the components listed in Figure 2, including a Spring Boot backend and an Angular frontend with two-way binding. When requirements change in the future, e.g., additional AFOLU systems are added or modified, we adjust the textual class diagram notation, and another run of the code generator adjusts the data structure of the entire web application while keeping the hand-written parts untouched.

Of course, when calculations in the AFOLU projections change, we need to modify hand-written code, but generating the entire database structure and frontend files based on models lets the software engineer focus on domain-specific problems in the software system rather than repetitive and error-prone tasks such as updating database-related components after a change in the data structure. The software engineer, however, stays in full control over the system due to the TOP class-mechanism that lets engineers extend or even overwrite any generated file. While extending Java files is used to add functionality to the frontend or domain logic to the backend, overwriting was used in this project to customize the CSS file in the frontend.

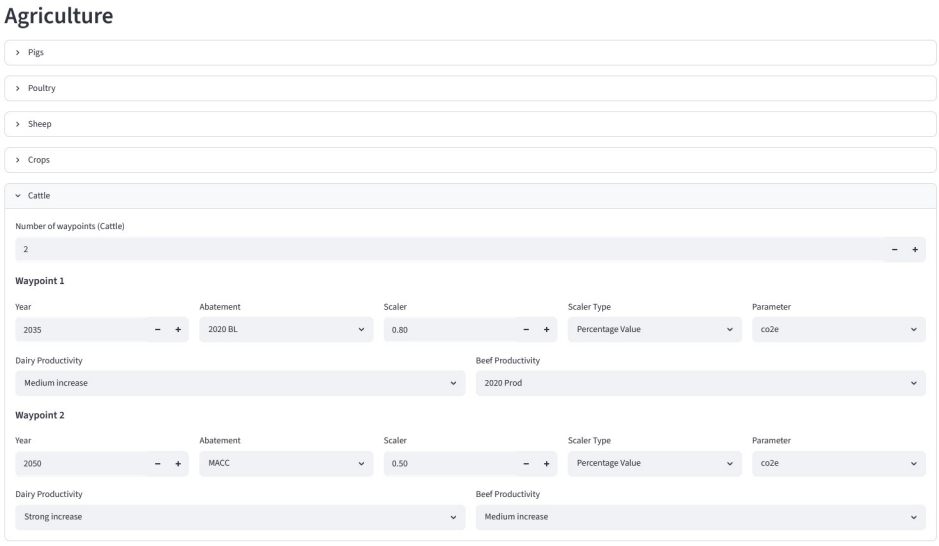

Figure 4: Section of the OptiGob 2.0 scenario configuration interface.

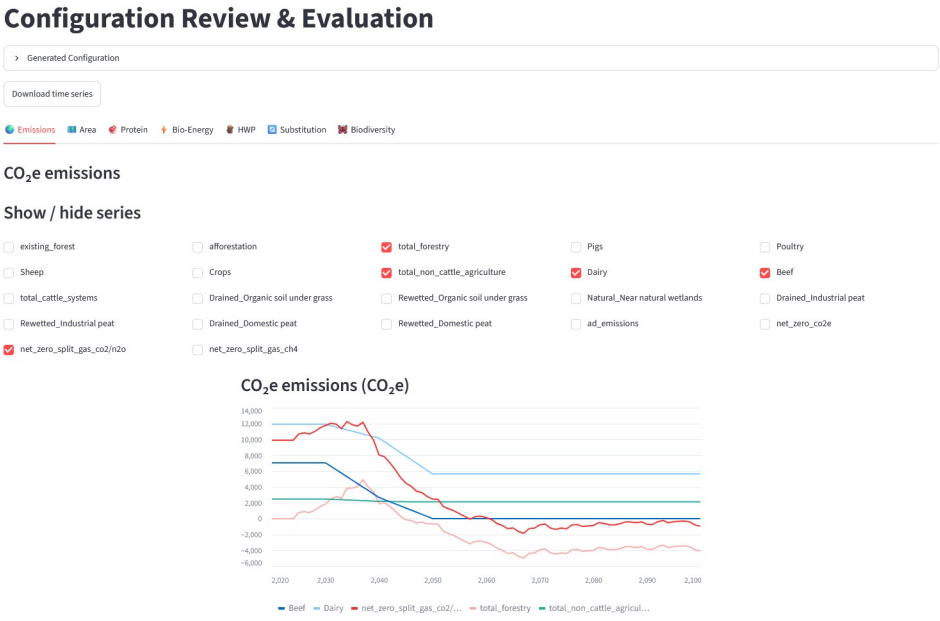

Figure 5: Section of the OptiGob 2.0 scenario evaluation interface.

5 THE SOFTWARE OPTIGOB 2.0

OptiGob 2.0, the first MDE version, supports a complex scenario configuration, a detailed calculation of AFOLU projections, and a comprehensive evaluation of those projections. In the following, we describe each of those components in detail.

5.1 Scenario Configuration

The central expansion in terms of functionality from OptiGob 1.0 [15] is the transformation from static configurations to a dynamic way point-based scenario configuration and evaluation.

Way point-based systems. Figure 5 shows one section of the configuration screen where we can see a selection of two way points for cattle. For each way point, we select a target year, the abatement type, a scaler value, a selection whether this scaler is used directly as a multiplier from the baseline value or as an absolute value, a selection by which parameter OptiGob will scale (i.e., by CO_2e emissions, area, or protein), and a level of productivity for both dairy cows and beef cows. Comparably, we can select way point for the other agriculture systems (i.e., pigs, poultry, sheep, and crops).

Similarly to agriculture systems, we can set way points for organic soil systems (i.e., organic soil under grass, industrial peat, and domestic peat). For each way point, we pick a year, and a rewetting ratio.

Static systems. Opposed to agriculture and organic soils, we do not configure way points for forestry and AD. *Forestry* is divided in existing forests, and afforestation. For both, users select the harvest rate (i.e., low or high) and whether we implement carbon capture and storage (CCS). Additionally, for afforestation, users configure an afforestation rate, and select ratios for the distribution of broadleaf and conifer, and organic and mineral soils.

Finally, users configure whether the *AD biomethane strategy* is implemented. If so, they can select an implementation year, and optionally CCS. In the configuration, they can also select additional grass biomethane as well as CO_2 removal via bioenergy with respective target years.

5.2 Running Scenarios

After configuration, the users run the OptiGob engine. In this engine, we first project the static systems. For forestry, there are values in the database according to each configuration that the engine scales up or down according to the afforestation rate. For AD strategies, there is data provided in the database that engine scales up or down according to the user input, and moves to the respective target years.

For organic soils projections, the database provides baseline parameters for drained and rewetted organic soils. The OptiGob engine runs iteratively through the way points, changes the parameters of the different soils according to the rewetting ratio and linearly connects the parameters in each data point according to the way point years. Additionally, the OptiGob engine creates a static timeline for near natural wetlands that are not directly configured by the user.

In the next step, the OptiGob engine projects non-cattle agriculture. The engine runs iteratively through the way points, and retrieves and scales data from the database according to the way point configuration. Since cattle has a significant impact on both greenhouse gas emissions and protein production of the entire AFOLU sector, the OptiGob engine additionally incorporates projections of other systems when calculating a cattle emissions budget. The appropriate dairy-to-beef ratio within cattle systems remains a subject of ongoing discussion among domain experts. OptiGob 1.0 lets the user set a fixed ratio, while version 2.0 reflects economic considerations. Afterwards, the engine fills this budget to create a data point for each way point, and linearly connects the parameters for each data point.

Finally, the OptiGob engine adjusts the projections in a process we call *area balancing*. When scaling systems up or down, we dedicate a different amount of area to this system. In OptiGob 2.0, we model these land use changes following this rule set: (i) in organic soils, we change areas from drained to rewetted areas, however, drained organic soil under grass areas can also be used for beef cattle or afforestation, (ii) when scaling cattle and sheep down, those areas can be dedicated for AD or afforestation, (iii) crop alternates between croplands and no-crop-croplands, and (iv) within areas, a subset is referred to as high nature value areas, we use those to assess biodiversity goals. So far, these land use changes are only modeled on a global scale for all of Ireland; future work will be concerned with mapping these changes from a global to local level by connecting different land use models.

5.3 Scenario Evaluation

After configuring a scenario and creating the projections with the OptiGob engine, the user can evaluate the scenario. **??** shows a part of the configuration screen for an example scenario. In multiple tabs for greenhouse gas emissions, area-use, protein production, bio-energy, harvested wood production, substitution credits, and biodiversity areas, the user can evaluate the AFOLU scenario from multiple analytical perspectives using timelines representing the different systems. Additionally, for greenhouse gas emissions, OptiGob calculates a timeline for all CO_2e emissions together as well as split into CO_2 plus N_2O and CH_4 to assess the performance for different net-zero goals.

In summary, OptiGob 2.0 enables the user to configure complex and dynamic AFOLU scenarios. Researchers and policy makers can set goals and scalers for different AFOLU systems at various points in time. OptiGob 2.0 produces projections for the various AFOLU systems and models their interactions to create timelines of the systems and their numerous parameters that researchers and policy makers can use to assess, e.g., whether Ireland's AFOLU sector achieves the EU carbon-neutrality goal by 2050.

6 DISCUSSION

Our engineering approach enables us to compare the hand-written and the model-driven engineering methods to create OptiGob 2.0. After that, we discuss the insights we gained during the development process, followed by a description of the maintainability and extensibility of model-driven OptiGob.

6.1 Differences Between the Hand-Written and the Model-Driven Software Version

Based on the descriptions of the functionality of both the hand-written and the MDE-generated software systems in the previous section, we can conclude that the differences in terms of functionality are mostly negligible. E.g., there are differences in scenario configuration input items due to slightly different mechanics between Streamlit and MontiGem's GUI DSL. The most notable difference is MontiGem's backend that allows multiple users to work from different clients on the same configuration, thus enabling concurrent multi-user access akin to collaborative editing environments.

When comparing the complexity of the hand-written and the model-driven version with respect to the overall hand-written lines of code, the difference is also only subtle. The hand-written version consists of 3,900 lines of code, the re-engineered version 3,700 lines of code before code generation. This near-equivalence reflects that the complexity of the domain logic itself did not change.

The differences become more clear when we look at the generated code: while the hand-written version includes a Streamlit frontend, domain logic, and a static SQLite database file, the MontiGem-supported version is a full-stack reactive web app including a Spring Boot backend, an Angular frontend with two-way binding, WebSocket commands, observer pattern, serialization, and

persistence with around 140,000 generated lines of Java, TypeScript, and HTML code. The MDE-driven approach improves not only the code quality, but also maintainability and extensibility.

6.2 Findings from the Engineering Processes

After an initial planning phase to assess the additional requirements for OptiGob 2.0, we chose an agile approach with regular meetings with domain experts to understand and implement OptiGob iteratively across individual systems and fields.

LLMs provide a valuable instrument for domain experts, in this case agri-sustrainability researchers, to build prototypes and establish shared understanding with software engineers regarding requirements and design expectations. However, a thorough analysis of the application domain from a *software engineering perspective* is necessary to apply software engineering principles. Without appropriate guidance, LLMs are currently not able to accomplish the appropriate abstractions, as it tends to oversimplify problems and thus limit extensibility and maintainability.

The agile approach was very valuable for confirming or revising progress at multiple stages of the development process. In order to build common ground, discussions were held in which software engineers explained the domain back to domain experts to validate their domain-specific knowledge; this is a form of knowledge validation through teach-back [26].

In the transformation process from hand-written code to MDE-generated code, we included Anthropic's Claude model Sonnet 4.6 into the process. Combinations of using LLMs with an MDE-based code generator were previously described as *vibe modeling* by Cabot [11]. We observed that Claude tends to oversimplify OptiGob's data structure from independently analyzing hand-written code. One example for this oversimplifying was a suggestion to merge fields and systems and, thus, exacerbating future expansions in the data structure. However, Claude was reliable when it came to transforming a visual representation of the class diagram (that was manually extracted from the hand-written version) into MontiGem's DSL for class diagrams. For the necessary modifications, it took regular iterations of exploring the problem with the AI agent and manually reviews in the progress to control the LLM support.

For creating the GUI models, Claude was able to follow the MontiGem GUI DSL documentation to transform the handwritten Streamlit pages into GUI models, as well as imitating the frontend design by overwriting the CCS file. For this frontend task, only little guidance from a human engineer was required.

For translating the domain logic from Python into a backend service, we followed this protocol iteratively: (i) the software engineer selects a modular section of the domain logic to translate, (ii) Claude translates the section into the Java backend service, and (iii) the software engineer reviews and tests the newly added section.

6.3 Maintainability and Further Extensions

Good maintainability of the system will continuously be important: datasets from sources such as the annually published national inventory report from Ireland's environmental protection agency will have to be updated, more AFOLU systems that are not yet included in OptiGob will be added.

While OptiGob 2.0 was improved in comparison to version 1.0 in terms of enabling the dynamic configuration of scenarios with way points and scalers, and a more detailed evaluation using timelines, as well as providing a maintainable and extensible foundation, there is already a need for additional features, e.g., multi-objective optimization and a connection to other analytical models. In more detail, it is time-consuming and inefficient for users to optimize towards various goals by manually changing configurations and re-running the OptiGob engine. Here, there is a need for an automated *multi-objective optimization* to discover scenarios that satisfy constraints from a diverse set of stakeholders. In addition, OptiGob operates only on a national level. While OptiGob supports investigations of how many emissions should be mitigated, it does not provide any insights on where exactly there could be a reduction of cattle or an appropriate area for afforestation. One solution could be a *connection to another low-level model* that takes geographical data into account.

Our MDE-based approach will raise the development speed for changes in the data structure and additional features. To demonstrate that, we can compare the necessary modifications in the software system for a newly introduced feature. In the hand-written version, we would need to implement a new frontend page, manually connect that to the also hand-written domain logic, and manually connect that to an updated database. With our model-driven approach, we would change the data structure in the class diagram, add a GUI model, and add domain logic. For the remaining steps, we would use the code generator. The model-driven code generator not only accelerates the development process, but also maintains high code quality. Thus, our new model-driven approach provides a structurally stable basis for upcoming changes and additions.

7 RELATED WORK

Using model-driven techniques to build research software from scratch is an established methodology in the literature. Combemale et al. present a selection of publications in this area [13]. In our work, however, we focus on a problem where an existing software system needs to be transformed into a model-driven system.

Various approaches use model-driven reverse engineering techniques to understand legacy code [29, 31]. The term *reverse-engineering* describes the process of analyzing a system, its components and their relationships, and creating models to represent them on a higher abstraction level. In this work, however, we go a step further by applying the stages reverse engineering, restructuring, and forward engineering (often referred to as *model-based or model-driven re-engineering* [12, 30]).

Lano et al. proposed a framework for re-engineering legacy projects using model-driven techniques [24]. They suggest agile model-driven re-engineering where they used a process consisting of (i) *abstraction*, (ii) *quality analysis & restructuring*, (iii) *extract specifications*, and (iv) *generate target platform code & tests* to re-engineer 100 cases. These cases included tasks such as translating VB6/VBA code to Python 3, translating COBOL code to Java and C#, or translating Python code to UML for a transformation to Java. Their findings include significant quality improvements, and a demonstration of over 80% translation accuracy. In a second paper, Lano et al. also use grammar-based transformation rules in a MDE

re-engineering problem on software sustainability (i.e., reducing the energy consumption) [25]. In comparison to our approach, we were able to keep the first three stages brief due to the recent development of the hand-written version, and focus on generating the target platform code. While Lano et al. used LLMs for documenting their code and transformation languages for translating their legacy code, we used LLMs also for translating tasks.

Another example for this category of problems was presented by Böhm et al. [8] where they migrated a software system at scale including 1,500 web pages and 3GB of code in a process they called model-driven migration. They also describe a similar process of (i) *analysis*, (ii) *enrichment*, (iii) *synthesis*, and (iv) *transition* to achieve a fully automated migration of an industry-scale legacy system to a model-driven software system. In contrast, we focused on domain-specific research software instead of industry scale, and while we used LLMs, they only mention it in their future work section.

8 CONCLUSION

In this paper, we present OptiGob 2.0, a model-driven software system to support agri-sustainability researchers in Ireland to configure AFOLU scenarios and evaluate their projections. Addressing *RQ1*, we not only applied software engineering practices to expand on the first version of OptiGob to include features such as dynamic way point-based scenario configurations and timelines for evaluations, but also, addressing *RQ2*, re-engineered this system to become a model-driven system.

This new system builds a solid foundation for the future tasks of OptiGob when it comes to maintainability and extensibility while simultaneously providing a full-stack reactive web application without raising the complexity from the hand-written Streamlit script.

In summary, we present a case where domain experts build research software using the potential of LLMs in a process of democratizing software development. As requirements grow, the complexity inherent in the evolution of such systems increases correspondingly. MDE represents a promising mechanism to support the expansion of research software development while maintaining high code quality.

ACKNOWLEDGMENTS

We would like to thank Colm Duffy and Takfarinas Saber for their continuous input during the development of OptiGob 2.0, as well as Alexander Hellwig, Sebastian Will, and Om-Preetam Valasa for their technical support in the re-engineering process with MontiGem.

REFERENCES

[1] 2018. Regulation - 2018/841 - EN - EUR-Lex — eur-lex.europa.eu. https://eur-lex.europa.eu/eli/reg/2018/841/oj/eng. [Accessed 26-06-2026].
[2] 2018. The Food, Agriculture, Biodiversity, Land-Use and Energy Consortium. https://fableconsortium.org/. [Accessed 26-06-2026].
[3] 2021. European Climate Law — climate.ec.europa.eu. https://climate.ec.europa.eu/eu-action/european-climate-law_en. [Accessed 26-06-2026].
[4] 2021. *Streamlit.* https://streamlit.io/ [Accessed 26-06-2026].
[5] 2022. Sectorial Emissions Ceilings — gov.ie. https://www.gov.ie/en/department-of-climate-energy-and-the-environment/publications/sectoral-emissions-ceilings/. [Accessed 26-06-2026].
[6] 2023. *Claude Code.* https://code.claude.com/docs/en/overview [Accessed 26-06-2026].
[7] Simon Anastasiadis, Suzi Kerr, Wei Zhang, Corey Allan, and William Power. 2014. Land use in rural New Zealand: spatial land use, land-use change, and model validation. (2014).
[8] Tobias Böhm, Jens Guan Su Tien, Mohini Nonnenmann, Tom Schoonbaert, Bart Carpels, and Andreas Biesdorf. 2026. Model-Driven Legacy System Modernization at Scale. *CoRR* abs/2602.04341 (2026). https://doi.org/10.48550/ARXIV.2602.04341 arXiv:2602.04341
[9] Constantin Buschhaus, Arkadii Gerasimov, Jörg Christian Kirchhof, Judith Michael, Lukas Netz, Bernhard Rumpe, and Sebastian Stüber. 2024. Lessons learned from applying model-driven engineering in 5 domains: The success story of the MontiGem generator framework. *Journal Science of Computer Programming* 232 (Jan 2024), 103033. https://doi.org/10.1016/j.scico.2023.103033
[10] Michael L. Bynum, Gabriel A. Hackebeil, William E. Hart, Carl D. Laird, Bethany L. Nicholson, John D. Siirola, Jean-Paul Watson, and David L. Woodruff. 2021. *Pyomo–optimization modeling in python* (third ed.). Vol. 67. Springer Science & Business Media.
[11] Jordi Cabot. 2025. Vibe Modeling: Challenges and Opportunities. In *Advances in Conceptual Modeling - ER 2025 Workshops, FCM, CMLS, LLM4Modeling, OntoCom, and QUAMES, Poitiers, France, October 20-23, 2025, Proceedings (Lecture Notes in Computer Science, Vol. 16190)*, Claudenir M. Fonseca, Anna Bernasconi, Sergio de Cesare, Ladjel Bellatreche, and Oscar Pastor (Eds.). Springer, 105–118. https://doi.org/10.1007/978-3-032-08620-4_7
[12] Elliot J. Chikofsky and James H. Cross II. 1990. Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Softw.* 7, 1 (1990), 13–17. https://doi.org/10.1109/52.43044
[13] Benoît Combemale, Jeffrey G. Gray, and Bernhard Rumpe. 2023. Research software engineering and the importance of scientific models. *Softw. Syst. Model.* 22, 4 (2023), 1081–1083. https://doi.org/10.1007/S10270-023-01119-Z
[14] Imke Drave, Arkadii Gerasimov, Judith Michael, Lukas Netz, Bernhard Rumpe, and Simon Varga. 2021. A Methodology for Retrofitting Generative Aspects in Existing Applications. *Journal of Object Technology* 20 (November 2021), 1–24. https://doi.org/10.5381/jot.2021.20.2.a7
[15] Colm Duffy, Daniel Henn, Takfarinas Saber, and David Styles. 2026. OptiGob: A decision support tool for Agriculture, Forestry and Other Land Use transitions. *J. Open Source Softw.* 11, 119 (2026), 9271. https://doi.org/10.21105/JOSS.09271
[16] Michael Felderer, Michael Goedicke, Lars Grunske, Wilhelm Hasselbring, Anna-Lena Lamprecht, and Bernhard Rumpe. 2025. Investigating Research Software Engineering: Toward RSE Research. *Commun. ACM* 68, 2 (Jan. 2025), 20–23. https://doi.org/10.1145/3685265
[17] Thomas MR Gérard, Sietze J Norder, Judith A Verstegen, Jonathan C Doelman, Stefan C Dekker, and Floor van Der Hilst. 2025. Trade-Offs and Synergies Between Climate Change Mitigation, Biodiversity Preservation, and Agro-Economic Development Across Future Land-Use Scenarios in Brazil. *Global Change Biology* 31, 8 (2025), e70418.
[18] Arkadii Gerasimov, Nico Jansen, Judith Michael, Bernhard Rumpe, and Sebastian Will. 2025. A Model-Driven Approach to Design, Generation, and Deployment of GUI Component Libraries. In *18th ACM SIGPLAN International Conference on Software Language Engineering (SLE'25)*. ACM, 57–70. https://doi.org/10.1145/3732771.3742713
[19] Arkadii Gerasimov, Judith Michael, Lukas Netz, and Bernhard Rumpe. 2021. Agile Generator-Based GUI Modeling for Information Systems. In *Modelling to Program (M2P)*, Ajantha Dahanayake, Oscar Pastor, and Bernhard Thalheim (Eds.). Springer, 113–126. http://www.se-rwth.de/publications/Agile-Generator-Based-GUI-Modeling-for-Information-Systems.pdf
[20] Malte Heithoff, Nico Jansen, Jörg Christian Kirchhof, Judith Michael, Florian Rademacher, and Bernhard Rumpe. 2023. Deriving Integrated Multi-Viewpoint Modeling Languages from Heterogeneous Modeling Languages: An Experience Report. In *16th ACM SIGPLAN International Conference on Software Language Engineering (SLE'23)*. ACM, 194–207. https://doi.org/10.1145/3623476.3623527
[21] Richard Hopkins and Kevin Jenkins. 2008. *Eating the IT elephant: Moving from greenfield development to brownfield.* Addison-Wesley Professional.
[22] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sung Hun Kim. 2026. A Survey on Large Language Models for Code Generation. *ACM Trans. Softw. Eng. Methodol.* 35, 2 (2026), 58:1–58:72. https://doi.org/10.1145/3747588
[23] Sarah K Jones, Adrian Monjeau, Katya Perez-Guzman, and Paula A Harrison. 2023. Integrated modeling to achieve global goals: lessons from the Food, Agriculture, Biodiversity, Land-use, and Energy (FABLE) initiative. *Sustainability Science* 18, 1 (2023), 323–333.
[24] Kevin Lano, Howard P. Haughton, Ziwen Yuan, and Hessa Alfraihi. 2024. Agile model-driven re-engineering. *Innov. Syst. Softw. Eng.* 20, 4 (2024), 559–584. https://doi.org/10.1007/S11334-024-00568-Z
[25] Kevin Lano, Shekoufeh Kolahdouz Rahimi, and Zishan Rahman. 2025. Using MDE to support sustainable re-engineering. *J. Object Technol.* 24, 2 (2025), 2. https://doi.org/10.5381/JOT.2025.24.2.A15
[26] Yaoli Mao, Dakuo Wang, Michael J. Muller, Kush R. Varshney, Ioana Baldini, Casey Dugan, and Aleksandra Mojsilovic. 2019. How Data Scientists Work Together With Domain Experts in Scientific Collaborations: To Find The Right Answer Or To Ask The Right Question? *CoRR* abs/1909.03486 (2019). arXiv:1909.03486 http://arxiv.org/abs/1909.03486
[27] MontiCore Project. 2026. *CD4Analysis.* GitHub. https://github.com/MontiCore/cd4analysis

[28] Remi Prudhomme, Brian Duffy, James Gibbons, Cathal O'Donoghue, Mary Ryan, David Styles, et al. 2022. GOBLIN version 1.0: a land balance model to identify national agriculture and land use pathways to climate neutrality via backcasting. *Geoscientific Model Development* 15, 5 (2022), 2239–2264.

[29] Claudia Raibulet, Francesca Arcelli Fontana, and Marco Zanoni. 2017. Model-Driven Reverse Engineering Approaches: A Systematic Literature Review. *IEEE Access* 5 (2017), 14516–14542. https://doi.org/10.1109/ACCESS.2017.2733518

[30] Francisco Javier Bermudez Ruiz, Jesús García Molina, and Oscar Díaz García. 2017. On the application of model-driven engineering in data reengineering. *Inf. Syst.* 72 (2017), 136–160. https://doi.org/10.1016/J.IS.2017.10.004

[31] Hanan Abdulwahab Siala, Kevin Lano, and Hessa Alfraihi. 2024. Model-Driven Approaches for Reverse Engineering - A Systematic Literature Review. *IEEE Access* 12 (2024), 62558–62580. https://doi.org/10.1109/ACCESS.2024.3394732

[32] Pete Smith, Mercedes Bustamante, Helal Ahammad, Harry Clark, Hongmin Dong, Elnour A Elsiddig, Helmut Haberl, Richard Harper, Joanna House, Mostafa Jafari, et al. 2014. Agriculture, forestry and other land use (AFOLU). In *Climate change 2014: mitigation of climate change. Contribution of Working Group III to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change*. Cambridge University Press, 811–922.

[33] Thomas Stahl, Markus Völter, Jorn Bettin, Arno Haase, and Simon Helsen. 2006. *Model-driven software development - technology, engineering, management*. Pitman. http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470025700.html

[34] David Styles, Daniel Henn, Colm Duffy, Kevin Black, and Andres Martinez-Arce. 2025. Structured foresight reframes risk around transitions towards sustainable and resilient farming in a volatile world. (2025). Preprint available at Research Square.