Tracing and Fixing Inconsistencies in Clone-and-Own Tabular Data Models

Nassim Bounouas, Mireille Blay-Fornarino, Philippe Collet

Tracing and Fixing Inconsistencies in Clone-and-Own Tabular Data Models

Nassim Bounouas
nassim.bounouas@univ-cotedazur.fr
Doriane - Université Côte d'Azur, CNRS, I3S
Sophia Antipolis, France

Mireille Blay-Fornarino
mireille.blay@univ-cotedazur.fr
Université Côte d'Azur, CNRS, I3S
Sophia Antipolis, France

Philippe Collet
philippe.collet@univ-cotedazur.fr
Université Côte d'Azur, CNRS, I3S
Sophia Antipolis, France

Abstract

Many data-intensive applications handle tabular data with more advanced structuring and processes than spreadsheets, enabling end-users to copy and adapt tabular data and processes to create new templates or datasets anytime. Recent research advances demonstrated that, in such clone-and-own scenarios, actions performed on the data structure, together with cloning and adaptation actions, can be captured within an operation-based model to prevent the drift of the internal tabular data model. However, this approach is limited by the assumption that each operation must maintain consistency regarding dependencies generated by the domain-specific languages that connect the observed and computed data.

To address this challenge, this paper first introduces an evolved operation-based model that is designed to capture inconsistent tabular data while keeping a fine-grained trace of what part of the model is inconsistent. We then define specific trace operations to either fix a dependency in a model or remove one if its creating process is no longer relevant to the user. These operations support high-level editing scenarios on the tabular data, which enables easily fixing the equivalent of a spreadsheet formula or a process statement, or making the user aware that some part of the model is inconsistent while it is cloned. Additionally, we report on a positive scalability experiment on the tracing of large tabular data models with inconsistencies.

CCS Concepts

• **Software and its engineering → Software product lines**; *Software configuration management and version control systems.*

Keywords

Tabular data, clone-and-own, variability management, operation-based modeling, model-driven engineering, agronomy

1 Introduction

Data-intensive applications can be used in many different domains (*e.g.*, for scientific computation [27], in information systems [39]). These applications mainly gather and process tabular data to handle complex and tailored operations that cannot be handled by spreadsheets because of their limited expressiveness [12] or their poor performance on large datasets [13]. While they maintain the basic and unavoidable functionalities of spreadsheets [9], these data-intensive applications have to also provide some change and adaptation means as flexible as the spreadsheet ones. Consequently, the tabular models and processes that are conceived within these applications and populated with data can be considered as a final product that can be copied, totally or partially, to be adapted at any time in the process and by different kinds of final users [10, 43]. As a result, any tabular model at any time can be considered as a template to create a new model, being a product, or a template, with some data being kept in it, with some processes linking the data being kept, removed, or adapted.

This clearly characterizes a form of *clone-and-own* process [21, 29, 32, 41] at the level of the model used by the applications. While these applications usually provide many facilities for the users to adapt and reuse the tabular data models, they do not keep track of what is cloned at a level detailed enough to be able to differentiate the data models or reason on them. Understanding what has been really changed on a cloned and evolved model is not easy and propagating changes is at least time-consuming, cumbersome, and impractical in many cases.

Facing these issues and the well-known negative effects on maintenance [29, 32, 47], we have recently introduced a partial solution [8] that is based on operation-based modeling [7]. It enables to control of the clone-and-own process by keeping a complete trace that captures all atomic operations on the tabular model together with the cloning operations. These latter operations describe when a product is derived from another, and what information from its model is extracted in the cloned result. While the captured model allows obtaining a whole variability-related history of the tabular models, this approach is limited by the assumption for each cloning or adaptation operation to maintain consistency in terms of dependencies between data. These dependencies range from spreadsheet-like formulas that depend on other data to compute a result to more sophisticated low-code or domain-specific languages that define user-tailored processes over several data inputs. In the first proposed approach [8], all dependencies must always be consistent between each operation, begin atomic actions or variability-related ones for cloning. This assumption directly hinders its application in real settings as dependencies should be made temporarily inconsistent so that the users can clone, edit, and adapt without any constraint. This is indeed typically the case in trial-based processes of the agronomy software of our industrial partner, where subjects are observed according to different criteria, with related criteria, observations, and specific computations such as ANOVA[1] variants.

To address these issues, this paper defines the concept of consistency about data dependencies and proposes an evolved operation-based model. This model is designed to identify and manage potentially inconsistent tabular data using a comprehensive set of operations. We also demonstrate that our proposed implementation can incrementally verify the consistency of operation-based models

[1] ANalysis Of VAriance: statistical method used to compare the means of multiple groups to determine if at least one group is significantly different from the others.

that capture cloning and adaptation. Additionally, we report on a scalability experiment involving the representation of real tabular data from an industrial variability-rich agronomy software that has evolved over many years. It shows that our approach of controlled clone-and-own can scale in this context.

The remainder of the paper is organized as follows. Section 2 introduces the context of clone-and-own in tabular data applications, discusses related work with the first operation-based model, and defines the problem statement. Section 4 describes our extension of the operation-based model to handle inconsistent dependencies in the tabular model. In Section 5, we present and discuss our two evaluations, including the implementation of scenarios in Prolog. Threats to validity and current limitations are discussed in Section 6, while Section 7 concludes this paper and discusses future work.

2 Motivations

2.1 Context

Data-intensive processes are present in many software systems [27, 39] in which they rely heavily on structured data organized in tables or similar formats. A spreadsheet is then the simplest and the most flexible tool to exploit these tabular data [9, 10, 43] with live coding accessible to almost all end-users. As their expressiveness is seen as low [12] and large data sets do not scale well [13], spreadsheets are often replaced by dedicated tabular data applications. These applications gather and process tabular data to handle complex tailored data organizations and operations while keeping the flexibility and adaptability of spreadsheets.

While their expressiveness is seen as low [12] and large data sets do not scale well [13], research advances in the field of spreadsheets have notably focused on smell detection in formula [26], but also on extracting structural components and groups to reveal the underlying semantics [11, 18–20]. This can also be observed in all approaches building on model-driven engineering to describe a model of the spreadsheet (*e.g.*, ClassSheet [22]), and mainly to reason from a spreadsheet on an inferred model [14–16] containing functional dependencies and formulas. Interestingly, to make a bidirectional transformation [16], Cunha et al. use a representation based on atomic operations [7] on the spreadsheet itself and its model representation in ClassSheet. We consider that our approach is similar but with a business-oriented representation of the tabular data and, above all, variability-related support.

Our work is set in the context of an ongoing partnership with an industrial company, Doriane, specialized in highly configurable tabular data applications with a main focus on agronomy. Their application aims to support breeders and seed companies that manipulate many forms of data and processes over several decades despite their inherent variability and specificity. As in almost all experimental settings, the challenge lies in managing this variability, which led to the initial use of spreadsheets by agronomic companies due to their flexibility and accessibility. Consequently, the Doriane application provides enhanced functionalities, such as more robust data processing capabilities and analysis of complex datasets with greater precision and flexibility. This software facilitates data gathering and experimentation through a shared platform with a generic GUI and a core engine that supports an open, tabular data model. This model is designed to be extensible and highly customizable to various user needs and specific processes. The customization extends from application engineers creating templates and examples to end-users, at multiple levels, tailoring these models in their daily operations.

In this paper, we will use an example from our industrial partner to illustrate the problem and our contribution. Let us consider an agronomic research institute that is conducting a long-term study on the impact of climate change on crop yields. The study involves multiple species of crops, each plant across various plots with different soil types, irrigation methods, and micro-climate conditions. The objective is to understand which irrigation method results in the best crop yields.

Each year, new criteria might be introduced, and/or existing ones might be modified or removed based on the previous year's findings, which may lead to a heteroscedasticity situation. The collected and observed criteria are various in this context: irrigation methods, fertilizer types, soil conditions, etc. When a data model is extracted and/or derived by the end-user, changes to the criteria can inadvertently lead to inconsistencies in how data is gathered and recorded, *e.g.*, if one trial modifies the criteria for water usage (increasing the range of recorded values) without adequately adjusting the criteria in other trials, the variability within these groups will differ, leading to heteroscedasticity[2]. These inconsistencies may also occur with formulas that link several data, criteria, or observations, just like a more complex broken spreadsheet formula, but they may also affect Domain-Specific Languages (DSL) provided by the Doriane application to build specific processes, such as a tailored ANOVA. In an ANOVA, changes in the input can easily introduce significant problems affecting the reliability and validity of the statistical conclusions (*e.g.*, impact on group means and variances, changes in statistical power, and ethical concerns).

2.2 Clone-and-own in tabular data models

Tabular data applications should offer flexibility comparable to the one provided by spreadsheets, preventing users from resorting to them as an alternative. This leads to users being able to add, remove, or modify any part of the underlying tabular data model through high-level actions of the application UI. Additionally, similar flexible changes can be applied to what defines processes, from spreadsheet-like formulas to DSL or other low-code programs that demand few programming skills to fit the different end-user profiles. Hence, the tabular models and processes developed within these applications, filled with data, can be viewed as finished products ready to be duplicated, either wholly or partially, for adjustment at any stage and by various end users [10, 43]. Every tabular model can then serve as a blueprint for generating a new model whenever needed, functioning either as a complete product or as a template, retaining certain data and processes while allowing for the addition, removal, or modification of others. This situation characterizes an actual *clone-and-own* approach [21, 29, 32, 41] at the level of the tabular model. The support of the clone-and-own mechanism naturally becomes an unavoidable functionality of tabular data application since it empowers incremental evolution and allows any end-user to start from a template [8].

[2]Heteroscedasticity happens when the standard deviations of a predicted variable, observed over different values of an independent variable, are non-constant.

The clone-and-own approach is essential in research and data management for adapting products to suit different analysis scenarios or target audiences. This method allows for the replication of existing products, which are then modified to incorporate new processes or findings. This need can be easily illustrated, in our agronomic experiment field by considering two roles: a breeder, who is responsible for the scientific program, and a field worker, who is in charge of implementing this program. The breeder is designing a program assessing the growth and yield responses of tomatoes to various fertilizer regimes. She works incrementally by selecting and aggregating various criteria that are essential for observing and understanding the impacts of different fertilizers. This incremental design phase leads to the creation of a tabular data product, which serves as a tool for capturing and analyzing the impact on tomato plants.

This product illustrated in Figure 1 includes the various criteria that are essential for understanding the relationship between fertilizer application and plant growth outcomes. These criteria are evaluated through structured observations, where each observation specifically assesses a criterion and explicitly references the associated subjects, ensuring comprehensive data integration and traceability. The "criteria" here, refer to the specific aspects under observation, such as types of fertilizer or plant growth metrics. The "subjects" are the individual plants in a specific plot being observed. The "observations" are the collected data on a specific "subject" about a specific "criterion". This structured approach makes it easier to evaluate how different factors affect agricultural results, helping the company gain detailed insights and understanding. The concepts used from criteria to subjects also help in staying in the business model of the users, which is not the case inside a spreadsheet.

After the design of the trial experimentation by the breeder, its practical application begins. In this phase, the field workers leverage the designed trial framework by employing a clone-and-own strategy. They clone the original trial setup and make necessary adaptations tailored to the specific conditions and variables encountered in the field. This approach ensures that the foundational design created by the breeder is preserved while allowing for the flexibility required to address practical conditions.

When an agronomic company faces legal regulations, such as those controlling the use of phytosanitary products, the ability to adapt and tailor their processes becomes crucial. To comply with these legal requirements, the company must conduct parallel trials that focus specifically on regulatory interests. By cloning the existing trial setups, they can create specialized versions tailored to meet these regulatory standards without having to design entirely new trials from scratch. This enables the integration of new, legally mandated criteria and protocols within the existing trial framework, not only by adding specific requirements but also by refining the scope of the trials to cater precisely to regulation and targeted audiences.

2.3 Usage scenarios

To illustrate the problem to be tackled, we scope it by defining several usage scenarios that describe the types of modifications or reasoning that should be supported by the application on its tabular model. Those scenarios are illustrated with our running example represented in Figure 1.

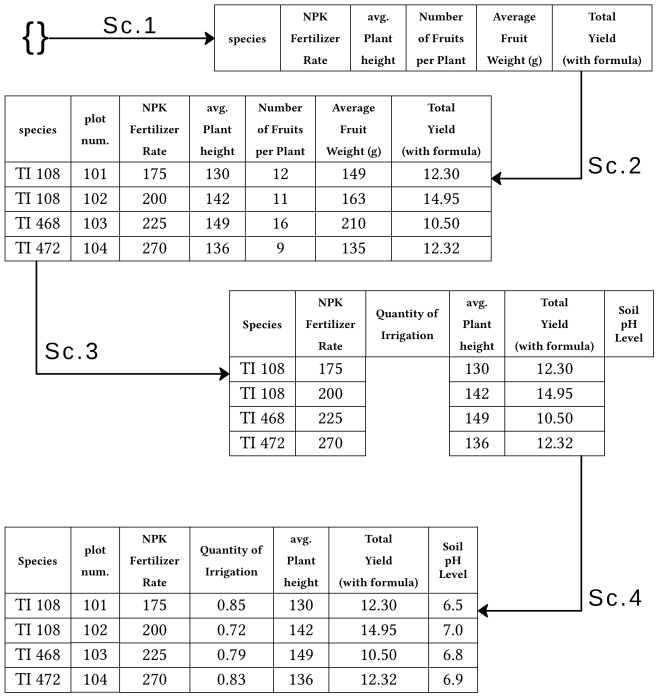

Figure 1: Illustration of clones and adaptions of tabular data model in agronomic trial management

Sc.1 As a Breeder, I want to design a trial to evaluate Tomato Growth and Yield responses to different fertilizer regimes. This trial should determine the impact of the quantity of fertilizer on plant height, fruit number, and total yield.

Sc.2 As a Field Worker, I want to clone the trial designed by the breeder to collect data on plant height, fruit number, and yield for each plot. This will involve adding a unique plot number to each data entry to ensure precise tracking and analysis of the treatment effects on tomato growth.

Sc.3 As a Breeder, I want to reuse and modify my first design to conduct a parallel trial assessing the impact of fertilizer regimes on soil acidity and its compliance with European phytosanitary regulations. It involves limiting the criteria to ensure compliance without compromising the data already collected. It is crucial to maintain traceability across both trials to ensure reliable analysis.

Sc.4 As a Field Worker, I want to reuse the modified design to realize the observations required by phytosanitary regulations. Re-associating results with their respective plot numbers is essential for maintaining traceability and compliance. This ensures that each data point can be accurately traced back to its origin, supporting regulatory audits and compliance checks that require detailed documentation of agricultural practices.

While tabular data applications usually provide facilities for the users to adapt and reuse their models, they do not keep track of what is cloned at a level detailed enough to differentiate the data models or reason on them. In the following, we will discuss related work and introduce the operation-based model we previously proposed [8] to tackle these issues. We will study its capabilities and limitations before introducing our contribution as an extension of this model.

3 Problem statement

3.1 Related work

Clone-and-own. The clone-and-own approach [21, 29, 32, 41] mainly consists of copying existing artifacts from a product to another while modifying them as needed to fit the new product's needs. It has been shown that clone-and-own may easily lead to unintentional divergence [29] between the products over time, creating in a form of model drift [47]. This drift in the models appears when different cloned product models become so different from each other that it is difficult or impossible to keep some consistency across all products. Change propagation, domain analysis, and quality assurance then demand an increasing and very costly effort.

Clone-and-own is generally avoided upfront and there have been numerous efforts to transition from it to the complete SPL paradigm [3]. The primary challenge lies in identifying features [3, 17, 36] within existing artifacts and cloned variants [48], often requiring prior knowledge of the feature set and yielding imprecise results [23, 25, 28, 34, 37, 38, 49]. Despite its prevalent use in industry [5], moving to a full SPL with feature management and mapping can be risky and expensive [31], particularly in scenarios with few products [40] or uncertain product evolution [21], potentially resulting in reduced flexibility[32]. Based on the classification of Antkiewicz et al. for the adoption process toward a full SPL [2] we have already suggested addressing this challenge by enhancing management support for the clone-and-own approach "with Provenance" with improved control and automation [8]

Managing clone-and-own product lines. Various approaches have been proposed to manage variability in clone-and-own product lines, leveraging feature traces for change propagation and product composition [24, 29]. Some methods utilize version control systems to identify features and variants, transitioning from code-based to feature-based reasoning [41, 42]. While variant identification is crucial, features are absent, posing challenges in representing differences between clones. A recent railway domain report highlights the significance of model divergence in model-driven SPLs, emphasizing the need for differencing large models between products and platforms [47]. Authors employ semantic lifting of model differences, aided by high-level change patterns derived from model repositories, for effective change propagation [45, 46]. Our context shares similarities, being model-driven and addressing model differences and drifts between evolving products. However, our focus lies more on representing and tracing model differences with inconsistencies, rather than mining them from external artifacts.

Operation-based modeling. Managing model differences is crucial in Model-Driven Engineering for tasks such as comparison, versioning, and model transformation [1, 6, 30, 33, 44]. Blanc et al.'s work on operation-based modeling involves representing models through sequences of elementary construction operations (*e.g.*, create, add, setProperty, setReference), enabling consistent detection and resolution of structural inconsistencies across models with different meta-models [7, 35]. This approach has been applied in the SPL community to identify features in product variant source code or UML models, always with the goal of migrating towards a unified platform within a complete SPL approach [4, 37, 49].

The next section focuses on our previous proposal of operation-based modeling to trace business actions and variability-related operations.

3.2 Tracing variability with operation-based modeling

In our previous work [8], we proposed an operation-based model to facilitate the cloning and evolution of underlying tabular data models in data-intensive applications. The notion of "observation" is the cornerstone of this operation-based model and enables the representation of what may typically be associated with a cell in a tabular model. A spreadsheet cell often lacks explicit characterization and relies instead on the implicit semantics suggested by its position as a column header, the first cell(s) in a row that describe what is observed in the row, or a cell evaluating something mentioned in its column header. On the contrary, our model introduced explicit definitions of these notions, represented by the concepts of observation, subject, criterion, requirement and dissociating the value from the observation it values. This model is illustrated in Figure 2 with its modification or addition appearing in red.

The observation is the cornerstone of the model. It serves both as a generic element that facilitates the representation of subjects and criteria and as the concrete representation of data collection. This concrete observation pertains to a subject that describes the object under consideration and evaluates an aspect of it, which is represented as a criterion.

The requirement is not an observation itself but a reference to a set of criteria applied to a criterion and transitively to the observations valuating it. This concept embodies the representation of references, similar to how formulas express relationships in a spreadsheet. However, unlike formulas in spreadsheets, which primarily express relationships leading to computations, this concept allows us to capture and represent relationships in spreadsheets that are implied or known only to the spreadsheet's author. Observations are aggregated in a product, which is a cohesive collection of observations that encapsulates a specific instance of the data model. The concept of product is intrinsically linked to the concept of trace. The trace is a detailed record of actions, such as adaptations, extractions, and clones, that have been applied to the products over time to create new products. This linkage between products allows for a comprehensive understanding of how the product has evolved, capturing each step in its development and adaptation.

(1) [...]
(2) $createObservation(null, \{\}) \rightarrow SpeciesRef$
(3) $valuateObservation(SpeciesRef, "species")$
(4) $createObservation(null, \{\}) \rightarrow NPKFertilizerRateRef$
(5) $valuateObservation(NPKFertilizerRateRef, "NPKFertilizerRate")$

(6) $createObservation(null, \{\}) \rightarrow AvgPlantHeightRef$
(7) $valuateObservation(AvgPlantHeightRef, "avg.Plantheight")$
(8) $createObservation(null, \{\}) \rightarrow NumFruitsPerPlantRef$
(9) $valuateObservation(NumFruitsPerPlantRef, "NumberofFruitsperPlant")$
(10) $createObservation(null, \{\}) \rightarrow AvgFruitWeightRef$
(11) $valuateObservation(AvgFruitWeightRef, "AverageFruitWeight(g)")$
(12) $createObservation(null, \{\}) \rightarrow TotalYieldRef$
(13) $valuateObservation(TotalYieldRef, "TotalYield(withformula)")$
(14) [...]

Trace 1: First Product Design Trace (Figure 1).

The initial version of the model we proposed already supports specific scenarios such as Sc.1 and Sc.2 as shown in traces Trace 1 and Trace 2. Scenario Sc.1 involves the creation of a template from a product, establishing a baseline for future operations, while Scenario Sc.2 focuses on the use of such a template to initiate a new trial, incorporating adaptations at the product level as required by specific use cases. Unlike the two first scenarios, scenarios Sc.3 and Sc.4 present challenges that are not currently supported by this model as it did not support temporarily inconsistent products, such as the third product in Figure 1 with a loss of the plot criterion required by the measurement criteria. This situation can typically occur during the process of adaptation and refinement of the products designed to be used as templates (designed to be cloned, adapted, and reused). In the model previously proposed, all operations carry pre and postconditions that ensure that the products are always consistent before and after each atomic operation. Using these operations of the initial model version to support Sc.3 and Sc.4 would inevitably lead to the loss of essential criteria and observations, or would require the removal of key associations defining the trial framework (*i.e.*, the requirements placed on criteria, and the connections between subjects and their respective observations). This would directly compromise the structural integrity and the intended purpose of the trial setup. In addition, these two scenarios require that the tabular data application not only supports inconsistencies within these products but also empowers users to identify and rectify these inconsistencies effectively.

(1) [...]
(2) $createObservation(null, \{\}) \rightarrow PlotNumRef$
(3) $valuateObservation(PlotNumRef, "PlotNum.")$
(4) $createReq(NPKFertilizerRateRef, SpeciesRef, PlotNumRef$
(5) [...]
(6) $createObservation(SpeciesRef, \{\}) \rightarrow Species101Obs$
(7) $valuateObservation(Species101Obs, "TI108")$
(8) $createObservation(PlotNumRef,) \rightarrow Plot101Obs$
(9) $valuateObservation(Plot101Obs, "101")$
(10) $createObservation(NPKFertilizerRateRef,) \rightarrow NPK101Obs$
(11) $valuateObservation(NPK101Obs, "175")$
(12) $createObservation(AvgPlantHeightRef,) \rightarrow Height101Obs$
(13) $valuateObservation(Height101Obs, "130")$
(14) $createObservation(NumFruitsPerPlantRef,) \rightarrow Fruits101Obs$
(15) $valuateObservation(Fruits101Obs, "12")$
(16) $createObservation(AvgFruitWeightRef,) \rightarrow Weight101Obs$

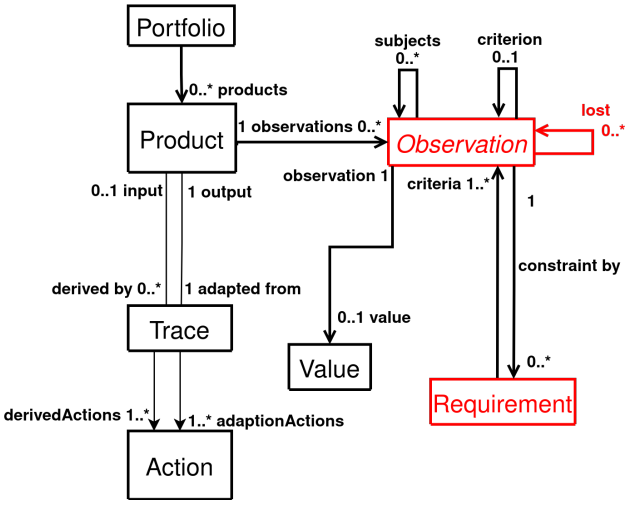

Figure 2: Enhanced model to capture and trace the clone tabular data models

(17) $valuateObservation(Weight101Obs, "149")$
(18) $createObservation(TotalYieldRef,) \rightarrow Yield101Obs$
(19) $valuateObservation(Yield101Obs, "12.30")$
(20) [...]

Trace 2: Second product trial completion trace (Figure 1).

To address these limitations, we consider it necessary to adopt a more flexible approach that allows for the capture and management of temporarily inconsistent products while keeping the trace mechanism relying on the operation-based technique. This technique provides fine-grained traceability and accountability in cloning and adaptation of processes in tabular data applications. By preserving the operation-based nature of the model, we expect that each step of data manipulation remains transparent, traceable, and reversible, allowing for more controlled data handling practices. In the following, we will describe how the model evolved to trace inconsistent products, but also how inconsistencies can be fixed afterward.

4 Contribution

To tackle the problem described in the previous section, we propose to keep the underlying operation-based model from Bounouas et al. [8] while extending it to represent inconsistent products (*cf.* Section 4.1, to trace those inconsistencies (*cf.* Section 4.2), and finally to be able to fix them (*cf.* Section 4.3).

4.1 Well-formed but inconsistent products

To support the different scenarios and the introduction of inconsistencies, we characterize **damaged** observations and requirements as model elements that reference lost observations, i.e., observations present in a previous product but removed during extraction, regardless of whether they are criteria or subjects.

Example: In the second and third products of our running example, the breeder decided to extract the criteria, subjects, and observations from the second product necessary to conduct her parallel trial on the impact of fertilizer on soil acidity. This extraction is guided by

her business perspective. However, the second product was not only completed by the Field Worker but also modified to associate each observation with a plot number. Consequently, the plot number criterion became a requirement for many other criteria, leading the extraction to potentially generate inconsistencies.

4.1.1 A model supporting lost observations. Our goal is to empower users to own the new product, even if it contains references to lost observations while providing them the means to fix these issues later. Given the large number of observations, the objective is not to assess inconsistencies retrospectively but to identify them along the extraction process. To maintain the integrity of the well-formed product, we construct transient observations that preserve the link to the lost observations.

According to our previous work [8], a **portfolio** is a set of products. A *product* is a set of observations along with the trace of operations that result in the product. An *observation* o is defined by a *value* v_o, which is a reference to a datum, the *criterion* c_o it evaluates, the set of observed *subjects* s_o, the *requirement* r_o it should conform to. We note an observation $o = (c_o, s_o, r_o, v_o)$ while eponymous functions are defined, such as $subjects(o) = s_o$. We note the absence of value by *null* and a place with any value by _. A *requirement* is designed as a set of criteria it refers to. A *criterion* is an observation c that neither evaluates any other criterion nor observes any subject.

A requirement $r_0 = req(c, c_1...c_n)$ constrains the observations on a target criterion c relative to their subjects associated with criteria $c_1...c_n, target(r_0) = c; subjects(r_0) = \{c_1...c_n\}$ For instance, the yield is calculated based on the average weight of the fruits and the fertilizer rate. The associated requirement is thus defined as req($Yield$, $\{FertilizerRate, AverageWeight\}$).

To support the concept of lost observations, the definition of a product has been extended to include a collection of observations, a specified set of lost observations, and the interrelationships between them.

A **product** p is then defined by:

- a *name* n_p
- a set of *observations* O_p
- a *trace* t_p that stores the actions whose product is the result (see next section).
- a set of *lost* observations L_p

We note it $p = (n_p, O_p, t_p, L_p)$ with again eponymous functions being defined, such as $lost(p) = L_p$. By construction, an observation belongs to a single product in a portfolio pf, *i.e.*, $\forall p_1 \in pf, \forall o \in p_1 \Rightarrow \nexists p_2 \in pf, o \in p_2$
It is thus possible to know for an observation which product aggregates it: $product(o) = p \in pf, o \in observations(p)$

We complete this definition with the following derived functions, which characterize a product's damaged observations and requirements:

An observation is damaged if its criterion or one of the subjects to which it relates is a lost observation:

$$damagedObservations(p) = \{o \in O_p \mid criterion(o) \in lost(p) \vee \exists s \in subject(o) \bigcap lost(p)\}$$

A requirement is damaged if at least one criterion contained in it is lost:

$$damagedRequirements(p) = \{r_o \mid \exists o \in O_p \mid r_o \in requirement(o) \wedge \exists c_o \in lost(p) \cap (subjects(r_o) \cup \{target(r_o)\})$$

4.1.2 Identifying Inconsistencies in a Product. Thanks to the previous definitions, the identification of inconsistencies in a product is straightforward. If a product does not reference any lost observations, then it is considered consistent. Conversely, if it contains lost observations, then it is considered inconsistent, meaning the union of the sets of damaged observations and requirements is non-empty.

4.2 Tracing the inconsistencies

To support the flexibility required for adapting a product, we define cloning actions that permit the occurrence of inconsistencies during the cloning process.

4.2.1 Adaptation Actions Supporting Inconsistencies. The creation of transient lost observations during the extraction phase requires defining an adaptation action *addLostObservation(product, lostObservation)* that is not directly accessible to the end-user and can only occur in a trace as a consequence of a cloning action.

$addLostObservation(product, observation) \rightarrow void$: Add a lost observation to a product

- Pre-conditions:
  1. the product and the observation exist and are well-formed,
  2. the observation to add is defined in another product.
- Post-conditions:
  1. the set of lost observations of the product contains the observation.

To enhance the end-user usability, we also define the composite operation *addFreeObservation*:

- $addFreeObservation(product, observation) \rightarrow void$: Add an observation $o = (c_o, s_o, r_o, v_o)$ to a product
  - Pre-conditions:
    1. the product and the observation exist and are well-formed,
    2. the observation is not already associated to a product.
  - Post-conditions:
    1. the product exists and contains the observation,
    2. $c_o \in observations(p) \bigcup lost(p)$
    3. $\forall s \in s_o, s \in observations(p) \bigcup lost(p)$
    4. $\forall c \in (subjects(r_o) \cup \{target(r_o)\})$, $c \in observations(p) \bigcup lost(p)$
    5. The trace includes the 'add' operations for the lost observations, whereas the atomic action itself is an 'addObservation'.

4.2.2 Free extraction. To improve the flexibility of our model and accommodate scenarios involving transitional inconsistent products, we introduce the *FreeExtract* operation. This operation allows for an under-constrained extraction based on a given product along

with a specified list of observations and requirements to be removed. It systematically constructs a trace to create a new product in which the cloned requirements or observations may reference observations that have been removed. This capability intentionally permits the generation of new products that may contain inconsistencies, providing users with the freedom to explore and reconfigure data relationships.

$$freeExtract(originProduct, observationsListToRemove, requirementsToRemove, newProductName) \rightarrow Product$$

This operation generates a new product that is structurally sound but may contain inconsistencies if it includes observations that have been designated as lost within the product.

Example:

(1) $freeExtract(p2, \{plotNum_{p2}, numFruits_{p2}\}, \{\}$ $"trialTwo") \rightarrow p3$
(2) $da : createObservation(null, \{\}) \rightarrow temp_{p3}$
(3) $da : addLostObservation(...)$
(4) $da : valuateObservation(temp_{p3}, "TotalYield")$
(5) $da : addObservation(p3, temp_{p3})$

Trace 3: The trace of the extraction from $p2$ leading to the product $p3$ illustrating the scenarios in Figure 1

The detailed process of a filtered extraction is as follows:

(1) Retrieve the source product's trace $a_1, ...a_n$.
(2) Remove every creation trace corresponding to an observation that is not kept. It includes removing valuation of these observations (*valuateObservation*).
(3) For each remaining *addObservation* action, apply an addFreeObservation:
   – For each observation about a non-kept criterion (*i.e.*, a criterion absent from the list of kept observations), add the criterion to the set of lost observations,
   – For an observation about a non-kept subject (*i.e.*, a subject absent from the list of kept observations), add the subject to the set of lost observations.

Therefore, we have a well-formed but inconsistent product in the sense that some of its observations refer to observations as criteria, subjects, or through requirements that were not cloned. Instead, the cloned observations refer to transient observations with the *lost* status and enabling the traceability of each original observation. Thus, all these referring observations are considered damaged. To fix this issue, it is necessary either to eliminate all references to these lost observations or to substitute some of them with new observations to fix the damaged relationship.

4.3 Fixing the inconsistencies

In the previous section, we introduced the operations that facilitate the identification of inconsistencies arising during the cloning and adaptation of products, generating damaged observations. These inconsistencies may stem from flawed requirements associated with criteria or from losses in the subjects and criteria referenced by the, now-damaged observations.

The subsequent discussion focuses on strategies and operations to support the incremental repair of damaged observations or requirements, thereby helping the user to progressively restore the overall consistency and functionality of the data model.

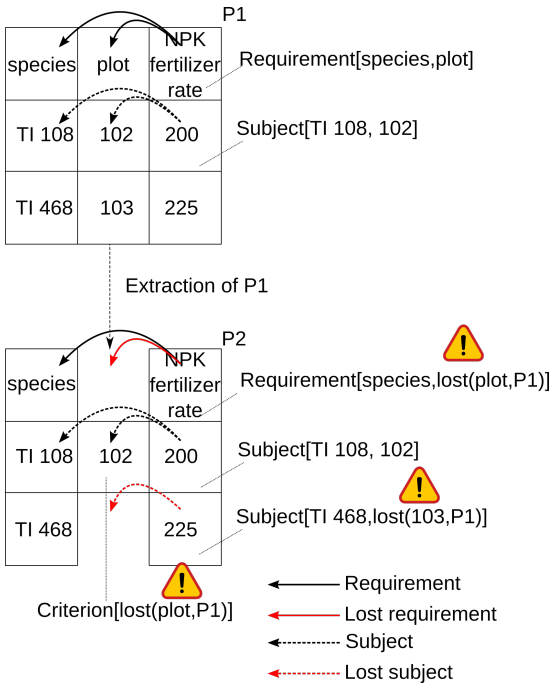

Figure 3: Extraction introducing inconsistencies

4.3.1 Illustrative scenarios. The extraction illustrated in Figure 3 can be formalized as follows.
For the first product $P1$:

$species_{p1} = (null, null, null, "species")$
$plot_{p1} = (null, null, null, "plot")$
$npk_{p1} = (null, null, req_{p1}, "NPKFertilizerrate")$
$req_{p1} = (npk_{p1}, [species_{p1}, plot_{p1}])$
$oti108_{p1} = (species_{p1}, null, null, "TI108")$
$o102_{p1} = (plot_{p1}, null, null, "102")$
$onpk200_{p1} = (npk_{p1}, [oti108_{p1}, o102_{p1}], null, "200")$
[...]

For the extraction leading to $P2$:

$species_{p2} = (null, null, null, "species")$
$npk_{p2} = (null, null, req_{p2}, "NPKFertilizerrate")$
$req_{p2} = (npk_{p2}, [species_{p2}, lost(plot_{p1})])$
$oti108_{p2} = (species_{p2}, null, null, "TI108")$
$o102_{p2} = (lost(plot_{p1}), null, null, "102")$
$onpk200_{p2} = (npk_{p2}, [oti108_{p2}, o102_{p2}], null, "200")$
$oti468_{p2} = (species_{p2}, null, null, "TI468")$
$onpk225_{p2} = (npk_{p2}, [oti468_{p2}, lost(o103_{p1})], null, "225")$

Following the extraction from Figure 3, several observations are damaged in *P2* because the criterion $plot_{p1}$ was not extracted even though there is still a requirement req_{p2} between npk_{p2} and $lost(plot_{p1})$. Indeed, the notion of plot does not seem immediately essential except perhaps in the case of the plot "102", since the observation $o102_{p1}$ has been cloned to be kept in $o102_{p2}$.

4.3.2 removeLostObservation. This operation addresses the need to handle the problem of a lost observation by asserting its deletion and cascading it to its references. It systematically removes the observation from the model and ensures that all links and dependencies that were previously referencing it are either updated or removed to preserve the integrity of the data structure. After its execution, the requirements depending on it are updated to remove the reference or if the observation was referenced as a subject, this subject is removed.

$removeLostObservation(product, observation) \rightarrow void$: remove a lost observation $o = (c_o, s_o, r_o, v_o)$ from a product p

- Pre-conditions:
  1. the product and the observation exist and are well-formed,
  2. the observation to remove is lost in p : $o \in lost(p)$
  3. if $c_o \neq null, \wedge c_o \in lost(p)$, $\nexists r \in damagedRequirements(p), c_o \in subjects(r)$
  4. Let the list of observations on o : $LO = \{o_i, o_i \in observations(p), criterion(o_i) = o\}$
- Post-conditions: There are no more references to o at the end of the repair process.
  1. $o \notin lost(p)$
  2. $\forall o_i \in observations(p)$, $criterion(o_i) \neq o \wedge \forall s \in subjects(o_i), criterion(s) \neq o$ $\wedge r_{o_i} = requirement(o_i), o \neq target(r_{o_i}) \wedge o \notin subjects(r_{o_i})$
  3. All observations that referred to the old criterion were removed : $\forall o \in LO, o \notin observations(p)$

Example: Fixing by removing a criterion. In the case of the second product illustrated in Figure 3, asserting the removal of the lost criterion $plot_1$ using $removeLostObservation(p2, lost(plot_{p1}))$ induces to remove all references to it.

All preconditions are verified and after the execution of the operation :

- no observation contained in LO is associated with $p2$;
- the requirement $req_{p2} = (npk_{p2}, [species_{p2}])$ does not reference $lost(plot_{p1})$ anymore ;
- any observation referencing $lost(plot_{p1})$ as its criterion has been removed *i.e.*, $o102_{p2}$;
- any reference to $o102_{p2}$ (or to an observation having the $lost(plot_1)$ as a criterion) as a subject is removed.

4.3.3 replaceFixCriterion. This operation addresses the need to handle the problem of lost criterion by replacing it. It serves as a corrective mechanism by allowing the substitution of a new criterion in place of a lost one.

Replacing a criterion with a new one requires ensuring that the model remains well constructed and that the requirements remain verified. Since we cannot correct lost observations (they have not been cloned), this operation is only possible if no lost observation refers to this old criterion. Furthermore, to ensure the consistency of the substitution, the new criterion can only carry a requirement that does not refer to any lost criteria.

$replaceFixCriterion(p, c_{lost}, c_{new}) \rightarrow boolean$:

Fix the references to the lost criterion c_{lost} by a new criterion $c_{new} = (null, null, r_{new}, v_{new})$. If by replacing all the references to the lost criterion, we have maintained consistency between all the observations, the operation is successful. If the new requirements cannot be verified, then the operation is not carried out.

Pre-conditions:

1. $c_{lost} \in lost(Product)$
2. $c_{new} \in observations(Product)$
3. $isCriterion(c_{lost}) \wedge isCriterion(c_{new})$
4. $\nexists o \in lost(p), criterion(o) = c_{lost}$
5. $\nexists s \in subjects(r_{new}), s \in lost(Product)$
6. Let the list of observations on c_{lost} : $LO = \{o, o \in observations(p), criterion(o) = c_{lost}\}$
7. Let the list of requirements whose subjects contains c_{lost} : $R = \{r \mid o \in observations(p), o = (\_, \_, r, \_), c_{lost} \in subjects(r)\}$

- Post-conditions: If consistency cannot be maintained, the system state will not be modified; otherwise, the following postconditions hold:
  1. $c_{lost} \notin lost(Product)$
  2. All observations that referred to the old criterion now refer to the new criterion and comply with its requirements : $\forall o \in LO, o \in observations(p), criterion(o) = c_{new}, conforms(r_{new}, o)$
  3. All requirements have been updated with the new criterion : $\forall r \in R, c_{new} \in subjects(r)$
  4. All observations comply with their requirements : $\forall r \in R, c_r = target(r)\ \forall o \in observations(p), criterion(o) = c_r, conforms(r, o)$

Example: Failure in the repair by replacing with a new criterion. If we add into Figure 3 a new observation which is the criterion defined as $plot_{p2} = (null, null, null, "LocalPlot")$ and we want to replace the reference $lost(plot_{p1})$ by a reference to $plot_{p2}$, the preconditions 1 to 3 would then be successfully checked. However the observation $o225_{p2}$ would not meet precondition 4, meaning that we should not be able to repair this observation and the requirements that involved $lost(p1, plot_{p2})$ as a subject, could not be satisfied. The operation would then not be applied.

4.3.4 replaceFixRequirement. Unlike the previous case where we prohibited the replacement of a criterion c_{lost} if there existed a lost observation referring to the criterion to be replaced, in the current

scenario, we automatically clone the observations associated with this requirement, which had not been cloned previously.

$replaceFixRequirement(p, r, c_{lost}, c_{new}) \rightarrow void$:
Fix the references to the lost criterion c_{lost} in requirement r, by a new criterion c_{new}
Pre-conditions:
(1) $target(r) = c_t, c_{lost} \in subjects(r)$
(2) $c_t \notin lost(p)$
(3) $c_{lost} \in lost(p), isCriterion(c_{lost})$
(4) $c_{new} \in observations(p), isCriterion(c_{new})$
(5) There are no other requirements in p that refer to c_{lost}
(6) Let the list of observations in p whose criterion we have to change for c_{new}
$L_{toFix} = \{o_s \mid o_s \in observations(p), criterion(o_s) = c_{lost}\}$
(7) Let the observations on c_t whose subjects refer to c_{lost}
$L_{toRepair} = \{o \in observations(p) \mid criterion(o) = c_t, o_s \in subjects(o), o_s \in lost(p), criterion(o_s) = c_{lost}\}$
– Post-conditions: All observations requiring fixing now refer to c_{new}. All necessary lost observations have been cloned and are referenced as subjects.
(1) $c_{lost} \notin subjects(r), c_{new} \in subjects(r)$
(2) $\forall o \in L_{toFix}, o \in observations(p), criterion(o) = c_{new}$
(3) $\forall o \in L_{toRepair}, \exists s \in subjects(o), criterion(s) = c_{new}$

Example: repairing a requirement. The requirement req_{p2} illustrates the application of this operation. We create a new criterion represented as $plot_{p2} = (null, null, null, "LocalPlot")$ and modify the fertilizer criterion so that it no longer refers to the previous criterion, but instead to the newly created one. Then, we repair req_{p2} substituting the reference from $plot_{p1}$ to $plot_{p2}$. A direct replacement of the former is not feasible because it includes a reference to plot 103, which was not cloned. Therefore, we choose to repair by fixing the requirement, so that it may potentially retrieve any lost observations and their clones.
In that case the preconditions are satisfied:

- the criterion $plot_{p2}$ exists and does not reference any requirement.
- the observation $o102_{p2}$ is contained within L_{toFix}.
- the observation $o225_{p2}$ is contained within $L_{toRepare}$.

After executing $replaceFixRequirement(p2, req_{p2}, plot_{p2}, plot_{p1})$, $p2$ corresponds to:
$o102_{p2} = (plot_{p2}), null, null, "102")$
$o103_{p2} = (plot_{p2}), null, null, "103")$
$req_{p2} = (npk_{p2}, [species_{p2}, plot_{p2}]$
$o200_{p2} = (npk_{p2}, [oTI108_{p2}, o102_{p2}], null, "102")$
$o225_{p2} = (npk_{p2}, [oTI468_{p2}, o103_{p2})], null, "225")$

5 Evaluation

5.1 First qualitative evaluation

Our first evaluation assesses the usefulness of our approach for practitioners on the basis of the current re-engineering of the application of our industrial partner, Doriane. The application is actually completely rewritten to be able to avoid the model drift of the previous version, which has been produced and evolved for more than 19 years now. As a first evaluation, we have then implemented an extension of our previous Prolog prototype to cover all scenarios and all operations described in Section 4. We have created examples that materialize the four scenarios and served also as unit tests. Then we validated the proposed scenarios, especially scenarios 3 and 4, with the main product owner (PO) of the new application, and one of the consultants, who have strong experience based on the company's historical products.

With the product owner, we have created example traces from our proposed model to illustrate scenarios 3 and 4. These traces use the demonstration data and experiments that are used by the PO and consultants to showcase the current Doriane application to future customers. Both scenarios have been validated by the PO while the tracing and fixing functionalities that can be implemented from this reasoning capabilities have been integrated into the new application backlog. The first minimum viable product of the tracing feature is under development at the time of writing.

The consultant we interviewed supports customers when they configure the Doriane application to their experimentation needs and when they migrate data into it. With him, we managed to take the last large reconfiguration that was made on a customer site to verify that the different modifications made can be captured by the model. As a result, all the changes made to the data organization and simple formulas have been seen as *able to be captured*. There were some difficulties with some analysis procedures that are currently hard-coded in the application and should be re-implemented with DSLs in the new version, but they were not related to the model capabilities.

5.2 Scalability evaluation

To evaluate the scalability of our model within real-world applications, we investigated the space and time consumption of our Prolog prototype when applied to large configurations. Based on data from Doriane's largest customers, we determined the average configuration size, which serves as the baseline for our scalability assessments. We created randomized products equivalent in size to the real products, ensuring that the randomized products are representative of the real ones. These products were generated using randomized cloning and extraction processes that selectively remove certain required observations.

We then conducted controlled experiments on a growing range of observations to compare three specific scenarios: the removal of all required observations, the removal of half of the required observations, and a full clone where every observation is retained. Those scenarios are encompassing the spectrum of customer behaviors and contexts encountered by Doriane. The full removal scenario reflects the practices of customers who extensively modify their processes and engage in exploratory research, whereas the full clone scenario represents those with stable and consistent research

processes. These customers conduct numerous trials but seldom revise their methodologies.

We are expanding the number of observations to assess the scalability of our model, focusing specifically on this metric rather than the number of products. This provides a more granular and insightful representation of customer activities. Large products are frequently divided into smaller ones to manage configuration complexities more effectively. Consequently, the capability to manage a few large products is often more critical than handling numerous smaller ones. Based on the measured memory footprint and time consumption we can derive several insights regarding the performance[3] of our model under these three situations (*cf.* fig. 4). There is a clear trend, as the number of observations increases, both the required memory and processing time escalate. However, the full clone situation where no observation is removed is clearly our worst case with the highest memory footprint and time consumed. This can be attributed to the trace of the cloned product that replicates every single observation, including their intricate tracing information, being fully recreated and re-executed, resulting in a heavier computational load and increased memory usage.

In contrast, the scenario with the maximal extraction — where all required observations are removed — shows the lowest consumption in both memory and time. This decrease can be explained by the reduced trace information necessary for the lost observations. Since these observations are not retained in the product, the system does not need to manage their full trace information, resulting in a lighter and more efficient process. The scenario with half of the observations removed logically falls between the two others, owing to a balanced load of trace information.

These results underline an important aspect of our model: it performs better in scenarios involving significant extraction. This is particularly advantageous in practical applications, as most of our partner's clients engage in extractions, with full clones without any extractions being less common. The model performance in large extraction scenarios suggests that our approach can handle extensive datasets effectively, making it suitable for contexts where tailored extractions and specific observations are essential.

6 Threats to validity and limitations

A primary concern regarding the construct validity of our model arises from the simplifications necessary to translate the clone-and-own processes into a computational framework. While our model has been refined through the partial proposal previously validated on some scenarios [8], it is also based on expert consultation and designed to reflect prevalent industry scenarios. Still, the subtleties of actual usage patterns may not be fully captured. We expect the incremental implementation of our proposal in Doriane's new application to provide validation and feedback on these patterns.

Internally, the prototype used for the scalability study shows some limitations as it is currently a proof of concept on which no optimization or profiling has been realized. Further work and its integration with the Doriane application should bring improvements. Besides, the study assumes a direct causal relationship between the model operations and the observed performance metrics. However, the presence of underlying, unaccounted-for factors within the tabular data and processes may disrupt these assumed relationships.

The external validity of our findings is closely tied to the contexts within which the model has been evaluated. The generated data, derived from scenarios provided by our industrial partner, raise questions about the model's applicability across a broader spectrum of domains or with datasets that differ substantially in size or complexity.

[3]The non-linear memory growth could be attributed to the Prolog engine's memory management tactics, including block-allocation and garbage collection strategies.

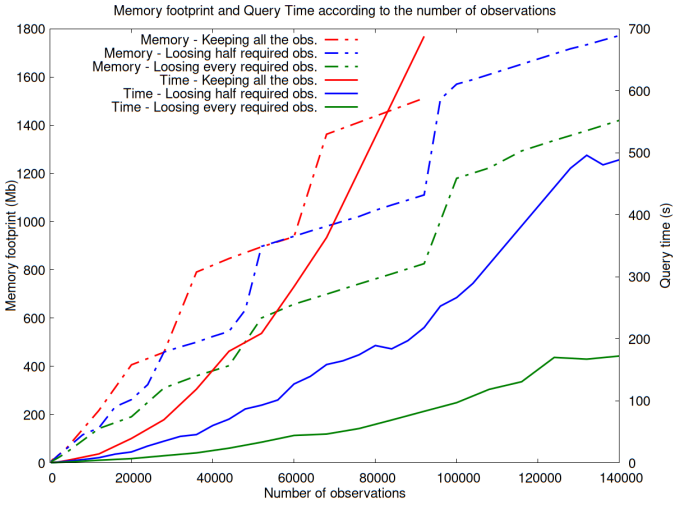

Figure 4: Space-Time comparative analysis

7 Conclusion

Operation-based modeling can be employed to trace all atomic business operations and cloning actions on the tabular data models used in data-intensive applications. This avoids model drift, but the current solutions could not represent inconsistent models, let alone the capacity to fix them.

In this paper, we have shown that our evolved operation-based model allows us to capture inconsistencies typically arising during the adaptation phase, which were previously unmanageable. This model robustly supports not only the process of identifying and tracing these inconsistencies but also provides mechanisms to fix them, thereby restoring coherence to the tabular data models. A first evaluation with practitioners shows that the scenarios covered by our proposal are highly relevant. A benchmark on scalability shows that a prototype implementation of our solution reasonably manages space and time efficiency.

Future work will focus on refining the model further, considering additional real-world scenarios and a complete coverage of formulas and DSLs. We also plan to extend the applicability of the approach to other domains where tabular data plays a critical role and where our industry partner could provide solutions with its new application integrating our proposal.

References

[1] Kerstin Altmanninger, Martina Seidl, and Manuel Wimmer. 2009. A survey on model versioning approaches. *International Journal of Web Information Systems* 5, 3 (2009), 271–304.

[2] Michał Antkiewicz, Wenbin Ji, Thorsten Berger, Krzysztof Czarnecki, Thomas Schmorleiz, Ralf Lämmel, Ștefan Stănciulescu, Andrzej Wąsowski, and Ina Schaefer. 2014. Flexible product line engineering with a virtual platform. In *Companion Proceedings of the 36th International Conference on Software Engineering*. 532–535.

[3] Wesley KG Assunção, Roberto E Lopez-Herrejon, Lukas Linsbauer, Silvia R Vergilio, and Alexander Egyed. 2017. Reengineering legacy applications into software product lines: a systematic mapping. *Empirical Software Engineering* 22, 6 (2017), 2972–3016.

[4] Wesley KG Assunção, Silvia R Vergilio, and Roberto E Lopez-Herrejon. 2020. Automatic extraction of product line architecture and feature models from UML class diagram variants. *Information and Software Technology* 117 (2020), 106198.

[5] Thorsten Berger, Jan-Philipp Steghöfer, Tewfik Ziadi, Jacques Robin, and Jabier Martinez. 2020. The state of adoption and the challenges of systematic variability management in industry. *Empirical Software Engineering* 25 (2020), 1755–1797.

[6] Enrico Biermann, Claudia Ermel, and Gabriele Taentzer. 2012. Formal foundation of consistent EMF model transformations by algebraic graph transformation. *Software & Systems Modeling* 11 (2012), 227–250.

[7] Xavier Blanc, Isabelle Mounier, Alix Mougenot, and Tom Mens. 2008. Detecting model inconsistency through operation-based model construction. In *Proceedings of the 30th international conference on Software engineering*. 511–520.

[8] Nassim Bounouas, Mireille Blay-Fornarino, and Philippe Collet. 2023. An Action-based Model to Handle Cloning and Adaptation in Tabular Data Applications. In *Proceedings of the 27th ACM International Systems and Software Product Line Conference-Volume A*. 201–212.

[9] Stefano Ceri, Piero Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, and Maristella Matera. 2003. *Morgan Kaufmann series in data management systems: Designing data-intensive Web applications*. Morgan Kaufmann.

[10] Yolande E Chan and Veda C Storey. 1996. The use of spreadsheets in organizations: Determinants and consequences. *Information & Management* 31, 3 (1996), 119–134.

[11] Zhe Chen and Michael Cafarella. 2013. Automatic web spreadsheet data extraction. In *Proceedings of the 3rd International Workshop on Semantic Search over the Web*. 1–8.

[12] Samuel Clemens. 2011. Five Ways To Tell You Have Outgrown Excel. https://www.insightsquared.com/blog/5-ways-to-tell-you-have-outgrown-excel/

[13] Rob Collie. 2012. Big Data is Just Data, Why Excel "Sucks", and 1,000 Miles of Data. http://www.powerpivotpro.com/2012/10/big-data-is-just-data-why-excel-sucks-and-1000-miles-of-data/

[14] Jácome Cunha, Martin Erwig, Jorge Mendes, and João Saraiva. 2016. Model inference for spreadsheets. *Automated Software Engineering* 23 (2016), 361–392.

[15] Jácome Cunha, Martin Erwig, and Joao Saraiva. 2010. Automatically inferring classsheet models from spreadsheets. In *2010 IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, 93–100.

[16] Jácome Cunha, João P Fernandes, Jorge Mendes, Hugo Pacheco, and Joao Saraiva. 2012. Bidirectional transformation of model-driven spreadsheets. In *Theory and Practice of Model Transformations: 5th International Conference, ICMT 2012, Prague, Czech Republic, May 28-29, 2012. Proceedings 5*. Springer, 105–120.

[17] Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. 2013. Feature Location in Source Code: A Taxonomy and Survey. *Journal of Software: Evolution and Process* 25, 1 (2013), 53–95. https://doi.org/10.1002/smr.567

[18] Haoyu Dong, Shijie Liu, Zhouyu Fu, Shi Han, and Dongmei Zhang. 2019. Semantic structure extraction for spreadsheet tables with a multi-task learning architecture. In *Workshop on Document Intelligence at NeurIPS 2019*.

[19] Wensheng Dou, Shi Han, Liang Xu, Dongmei Zhang, and Jun Wei. 2018. Expandable group identification in spreadsheets. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 498–508.

[20] Lun Du, Fei Gao, Xu Chen, Ran Jia, Junshan Wang, Jiang Zhang, Shi Han, and Dongmei Zhang. 2021. TabularNet: A neural network architecture for understanding semantic structures of tabular data. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 322–331.

[21] Yael Dubinsky, Julia Rubin, Thorsten Berger, Slawomir Duszynski, Martin Becker, and Krzysztof Czarnecki. 2013. An exploratory study of cloning in industrial software product lines. In *2013 17th European Conference on Software Maintenance and Reengineering*. IEEE, 25–34.

[22] Gregor Engels and Martin Erwig. 2005. ClassSheets: automatic generation of spreadsheet applications from object-oriented specifications. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. 124–133.

[23] Wolfram Fenske, Jens Meinicke, Sandro Schulze, Steffen Schulze, and Gunter Saake. 2017. Variant-preserving refactorings for migrating cloned products to a product line. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 316–326.

[24] Stefan Fischer, Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. 2014. Enhancing clone-and-own with systematic reuse for developing software variants. In *2014 IEEE International conference on software maintenance and evolution*. IEEE, 391–400.

[25] Stefan Fischer, Lukas Linsbauer, Roberto E Lopez-Herrejon, and Alexander Egyed. 2015. The ECCO tool: Extraction and composition for clone-and-own. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. IEEE, 665–668.

[26] Felienne Hermans, Bas Jansen, Sohon Roy, Efthimia Aivaloglou, Alaaeddin Swidan, and David Hoepelman. 2016. Spreadsheets are code: An overview of software engineering approaches applied to spreadsheets. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 5. IEEE, 56–65.

[27] Tony Hey. 2012. The Fourth Paradigm–Data-Intensive Scientific Discovery. In *E-Science and Information Management: Third International Symposium on Information Management in a Changing World, IMCW 2012, Ankara, Turkey, September 19-21, 2012. Proceedings*, Vol. 317. Springer, 1.

[28] Christian Kästner, Alexander Dreiling, and Klaus Ostermann. 2013. Variability mining: Consistent semi-automatic detection of product-line features. *IEEE Transactions on Software Engineering* 40, 1 (2013), 67–82.

[29] Timo Kehrer, Thomas Thüm, Alexander Schultheiß, and Paul Maximilian Bittner. 2021. Bridging the gap between clone-and-own and software product lines. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE, 21–25.

[30] Dimitrios S Kolovos, Richard F Paige, and Fiona AC Polack. 2006. Model comparison: a foundation for model composition and model transformation testing. In *Proceedings of the 2006 international workshop on Global integrated model management*. 13–20.

[31] Jacob Krüger and Thorsten Berger. 2020. Activities and costs of re-engineering cloned variants into an integrated platform. In *Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems*. 1–10.

[32] Jacob Krüger and Thorsten Berger. 2020. An empirical analysis of the costs of clone-and platform-oriented software reuse. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 432–444.

[33] Yuehua Lin, Jing Zhang, and Jeff Gray. 2004. Model comparison: A key challenge for transformation testing and version control in model driven software development. In *OOPSLA Workshop on Best Practices for Model-Driven Software Development*, Vol. 108. Citeseer, 6.

[34] Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. 2018. Variability extraction and modeling for product variants. In *Proceedings of the 22nd International Systems and Software Product Line Conference-Volume 1*. 250–250.

[35] Ernst Lippe and Norbert Van Oosterom. 1992. Operation-based merging. In *Proceedings of the fifth ACM SIGSOFT symposium on Software development environments*. 78–87.

[36] Roberto Erick Lopez-Herrejon, Sheny Illescas, and Alexander Egyed. 2018. A systematic mapping study of information visualization for software product line engineering. *Journal of software: evolution and process* 30, 2 (2018), e1912.

[37] Jabier Martinez, Tewfik Ziadi, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2015. Automating the extraction of model-based software product lines from model variants (T). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 396–406.

[38] Gabriela K Michelon, Lukas Linsbauer, Wesley KG Assunção, Stefan Fischer, and Alexander Egyed. 2021. A Hybrid Feature Location Technique for Re-engineering Single Systems into Software Product Lines. In *15th International Working Conference on Variability Modelling of Software-Intensive Systems*. 1–9.

[39] Celina M Olszak and Ewa Ziemba. 2007. Approach to building and implementing business intelligence systems. *Interdisciplinary Journal of Information, Knowledge, and Management* 2, 1 (2007), 135–148.

[40] Klaus Pohl, Günter Böckle, and Frank J van Der Linden. 2005. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer Science & Business Media.

[41] Julia Rubin, Krzysztof Czarnecki, and Marsha Chechik. 2013. Managing cloned variants: a framework and experience. In *Proceedings of the 17th International Software Product Line Conference*. 101–110.

[42] Julia Rubin, Andrei Kirshin, Goetz Botterweck, and Marsha Chechik. 2012. Managing forked product variants. In *Proceedings of the 16th International Software Product Line Conference-Volume 1*. 156–160.

[43] Christopher Scaffidi, Mary Shaw, and Brad Myers. 2005. Estimating the numbers of end users and end user programmers. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*. IEEE, 207–214.

[44] Matthew Stephan and James R Cordy. 2013. A Survey of Model Comparison Approaches and Applications. *Modelsward* (2013), 265–277.

[45] Christof Tinnes, Timo Kehrer, Mitchell Joblin, Uwe Hohenstein, Andreas Biesdorf, and Sven Apel. 2021. Learning domain-specific edit operations from model repositories with frequent subgraph mining. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 930–942.

[46] Christof Tinnes, Timo Kehrer, Mitchell Joblin, Uwe Hohenstein, Andreas Biesdorf, and Sven Apel. 2023. Mining domain-specific edit operations from model repositories with applications to semantic lifting of model differences and change profiling. *Automated Software Engineering* 30, 2 (2023), 17.

[47] Christof Tinnes, Wolfgang Rössler, Uwe Hohenstein, Torsten Kühn, Andreas Biesdorf, and Sven Apel. 2022. Sometimes you have to treat the symptoms: tackling model drift in an industrial clone-and-own software product line. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1355–1366.

[48] Yinxing Xue. 2011. Reengineering legacy software products into software product line based on automatic variability analysis. In *Proceedings of the 33rd International Conference on Software Engineering*. 1114–1117.

[49] Tewfik Ziadi, Luz Frias, Marcos Aurélio Almeida da Silva, and Mikal Ziane. 2012. Feature identification from the source code of product variants. In *2012 16th European Conference on Software Maintenance and Reengineering*. IEEE, 417–422.