



**HAL**  
open science

# An Action-based Model to Handle Cloning and Adaptation in Tabular Data Applications

Nassim Bounouas, Mireille Blay-Fornarino, Philippe Collet

► **To cite this version:**

Nassim Bounouas, Mireille Blay-Fornarino, Philippe Collet. An Action-based Model to Handle Cloning and Adaptation in Tabular Data Applications. SPLC '23: 27th ACM International Systems and Software Product Line Conference, Aug 2023, Tokyo, Japan. pp.201-212, 10.1145/3579027.3608991 . hal-04247084

**HAL Id: hal-04247084**

**<https://hal.science/hal-04247084v1>**

Submitted on 17 Oct 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# An Action-based Model to Handle Cloning and Adaptation in Tabular Data Applications

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 software systems require diverse data gathering and handling through processes that manipulate tabular data, often with a spreadsheet orientation. Variability in tabular data cannot be captured in a complete up-front analysis as everything is done at the final user level. She progressively adapts or clones some tabular data organized to conduct a process. Consequently these organized data are constantly both a final usable product and a potential candidate for cloning. This huge diversity, the high frequency of their evolution over time, and the intrinsic need to use cloning lead naturally to the usage of a clone-and-own approach with well-known negative impacts on maintenance and quality. In this paper we advocate that this can be replaced by controlling the clone-and-own process with provenance information that completely captures, at the domain level, the cloning actions and the adaptations applied on a product defining its clones. Each action over the process, its observations, and its data are captured in a complete model through traces of atomic adaptations, complemented with specific derivation and extraction actions. This model enables obtaining the whole history of both data and processes over time, as well as the accountability of variability-related actions. We report on a study showing the relevance of tackled problem in a variability-rich agronomy software of an industrial partner. We also show that a first prototype covers the extracted usage scenarios, from simple and entire cloning to more complex partial cloning.

## 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-gathering processes are central to many software systems, from scientific computation [30] to business intelligence for information systems [44]. These processes are relying on spreadsheets or software subsystems handling tabular data with an adaptable model for different sorts of tasks and many kinds of users [11, 48]. Even if the spreadsheet expressiveness is considered low [14] and that supporting software usually exposes poor performance on large data sets [15], their features are unavoidable in all data-intensive applications [10].

Being implemented through spreadsheets [29] or dedicated applications, those data structures and data-centered processes are naturally created and evolved through a *clone-and-own* approach [23, 33, 36, 46]. This process is inevitable as end-users are actually cloning and owning a final product (a spreadsheet or another form of tabular data), but at any time, it can be reused as a new template while adapting parts of the data organizations, keeping and removing parts of the data and relations between them. This is for example the case in all trial-based processes, where subjects are observed according to different criteria, with related criteria and observations being managed. At some point, end-users driving a trial can clone or adapt it (*e.g.*, by keeping some of the criteria, adding new ones) to reuse the trial in a new product, as this is at the core of their daily job.

In dedicated applications that provide an adaptable model of data handling, such as the one of our industrial partner, the clones and adaptations appear on all customer deployments. While the software itself focuses on providing adaptation and reuse capabilities over the data models, it keeps no track of what is cloned and how. Consequently, it is not possible to differentiate the data models between models at a customer site or between them and new template models developed by the application vendor. Reasoning on them or propagating changes is then a time-consuming or unfeasible task. Moreover, the underlying model is, as in many contexts [10, 11, 16–18, 48], completely spreadsheet-oriented with data only organized in columns and rows. Even with differencing tools, it is thus very hard to compare the models afterward to determine changes at the domain level, *e.g.*, a change in some criteria used to observe some subjects. This leads to the well-known negative impacts of clone-and-own on maintenance and quality [36], both on the developer teams and on the end-users on customer sites [33, 53]. However, migrating the software to a full Software Product Line (SPL) with a supporting platform [4, 19, 41] is not desirable here as the cloning capability on the tabular data model is essential to the customer’s everyday business.

In this paper, we advocate that this problem can be tackled by controlling the clone-and-own process with a complete trace that captures, at the domain level, both the cloning actions and the adaptations applied to a data model. We determine usage scenarios from interviews with our industrial partner, which develops a configurable agronomy software (*cf.* Section 2). As the actions on the model are domain-specific and come from end users, we rely on operation-based modeling [8] to first propose a model that tracks each action within a product, being a data model, under usage (*cf.* Section 3). The model is complemented with actions to describe when a product is derived from another, and what information

from its model is extracted in the cloned result. This allows to keep track of the cloning steps as well. Consequently, the model enables obtaining the whole history and accountability, which covers many usage scenarios with derivations and adaptations. We report on an experiment to analyze several data-intensive trial processes from customers of our partner for over 18 years, showing the relevance of these tackled scenarios (*cf.* Section 4). Together with its formal definition, we also provide a Prolog implementation of the model that demonstrates these usage scenarios on a small scale. Threats to the validity of our approach and its limitations are discussed in Section 5, while Section 6 concludes the paper and discusses future work.

## 2 MOTIVATIONS

### 2.1 Context

The context and validation of our work are conducted with an industrial partner, Doriane, which develops a highly configurable and adaptable experiment management software with a strong focus on agronomy. In the agronomy domain, breeding selection holds a prominent place. This process aims to improve the productivity and sustainability of a species' population by intentionally selecting and mating organisms to promote desired traits by producing new varieties. The breeder in charge of the selection is using every datum, phenotypic (the naked-eye observable characteristics), and/or genomic (the genetic characteristics observable in a laboratory), she considers as valuable to her work. These data cannot be standardized because of their multi-dimension variability. The collected pieces of information are specific to the type of species involved in the selection process, the targeted goal, but also to the breeder work methods and the company she works for. The targeted goal is dependent on the species and its market *e.g.*, improve the yield of a species or develop a resistance to a specific disease.

Very small agronomic companies start with spreadsheets to handle their data and processes as it is flexible and directly accessible to their breeders as end-users [48]. However, the lack of expressiveness and scalability of spreadsheets [14, 15] calls for dedicated software. Doriane currently meets this requirement through a configurable application to gather and exploit data and experiments. Doriane's domain engineers developed a common platform with a generic GUI and a core engine that handles an open model representing notably tabular data in a spreadsheet-oriented way (*cf.* Figure 1). The model, organized in lines and columns, is itself intended to be extensible and adapted to suit customer needs. Templates and examples of common trials are developed and inserted in the model and the platform by developers. Many adaptations are also made by the application engineering team when deploying at a customer site, mainly to create customer-specific processes, from the examples or from scratch. Finally, tailoring is also done by end-users along their daily work, from the breeder adapting and cloning to the field worker filling the tabular data, which could still adapt at the very last moment. It must be noted that, in this paper, we do not focus on formulas or expressions that can be used to relate dependent data. We concentrate on observations and criteria that constitute the backbone of manipulated tabular data in the targeted domains. Handling variability and evolution of formulas is a complementary important problem, which is part of our future work.

### 2.2 Problem statement

In our breeding illustration, as in many data-intensive processes, the breeder's job is made of complex activities that involve a lot of data and requires multiple associations to let her extract and infer enough knowledge to make the right research and development choices. Despite this complexity, the core of the job relies on vegetal species planted, observed, and evaluated on different criteria, with many dependencies between criteria and observations, all being managed through tabular data. In the following, we use a grasses<sup>1</sup> trial as an illustration, as it can easily show a combination of several observed subjects with multiple dimensions of study and dependencies.

An example of dependency is given in Figure 2. In the top part, the yield is dependent on the harvested species, the plot it grew in, and the harvest date. All those properties are aggregated in a so-called product, which matches the species' specificities and the breeder's requirements.

*Model adaptation and cloning at the end-user level.* The grass example might seem rather simple but is a true challenge because of its harvest cycle and the diverse targeted usages of grasses (herd feeding, ground cover of stadiums, erosion control, ornamental uses, biofuels...). Those different factors lead to a huge variability that is creating unmanaged clones at all levels. Moreover, these clones are spreadsheet models that encode the true domain variability (*e.g.*, a new criterion appearing to measure erosion) into changes over rows and columns. This cannot be easily managed in spreadsheets, even for small companies. Even with specific software such as the current Doriane solution, when the number of breeders, species and experimental fields grow, it becomes increasingly difficult to manage the different dimensions of variability.

While a very large part of the human-machine interfaces is common to many processes and many customers, most of the diversity and variability issues are then concentrated in the underlying model. A true domain model should convey some concepts of observations, criteria, and dependencies to capture the business semantics that is easily lost in spreadsheets [20]. But more importantly, a model instance is constantly adapted, maybe from a previous model instance, by both the developers in the software company and end-users on each customer site.

This leads to a kind of *model drift* [53] as there are divergences both between model versions of the Doriane software and the customer-adapted models, but also on customer sites when end-users may clone and adapt. This is currently managed by time-consuming propagations, which are, most of the times, partial, or not done at all as the differences between models and the origin of the clones are not known at all.

Furthermore, all end-users, such as breeders, only work at the product level, evolving their data management processes during a trial, or starting a more experimental trial from a previous one as a template. Agronomic companies are trying to standardize the processes and the data collected, but this standardization is restrained by the constant evolution of the trial methodologies, the data gathering and its analysis. An up-front domain analysis would then be too rigid here since the domain itself is evolving with new processes,

<sup>1</sup>The term used here is simplified. Grasses refers to *gramineae* and *poaceae* within the business lexicon.

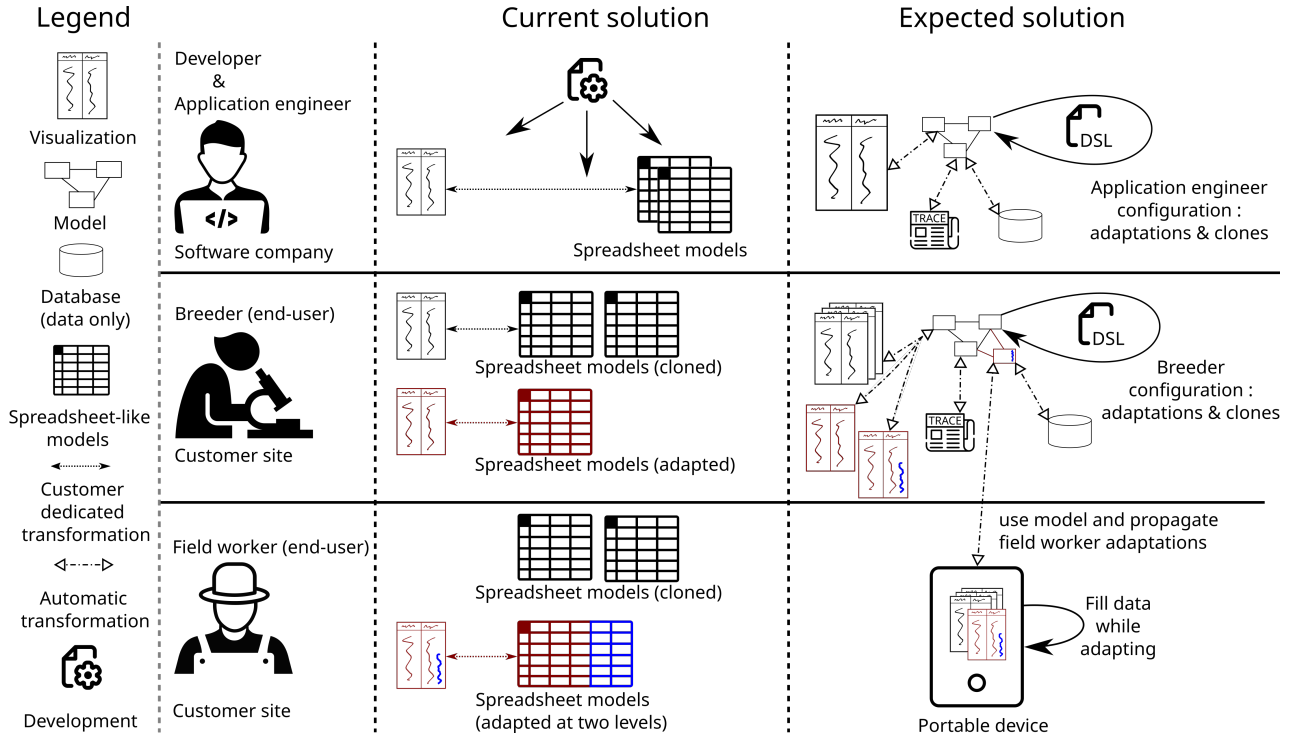


Figure 1: Adaptations and tailoring by different users

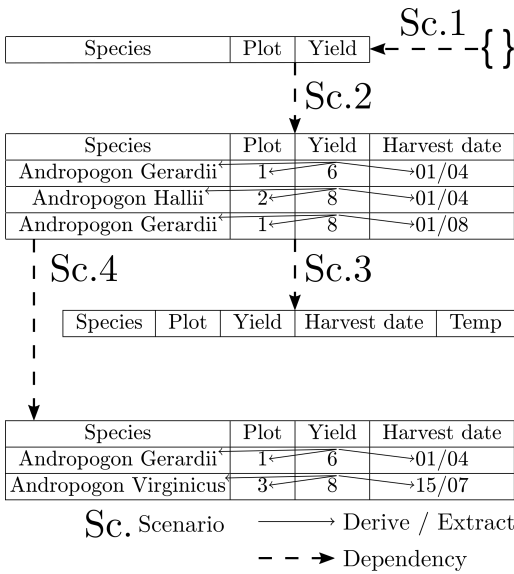


Figure 2: A flow of several cloning and adaptation scenarios

new data, and new products being developed, but even more problematic, everything is done at product-level with any product being a possible template at any time. We face here a typical situation of the *clone-and-own* paradigm with well-known negative impacts on

maintenance and quality [23, 33, 36, 46], but at a model level rather than a code level.

*Usage scenarios.* We conducted multiple end-user interviews whose goals were to determine the main sources of model drift within a product designed for a species and to characterize the adaptation and derivation support needed in this context. Our research team conducted face-to-face interviews with three customers of our industrial partner. A total of 10 individuals were interviewed: 3 breeders, 4 field workers, and 3 IT professionals. These interviews lasted approximately 90 minutes per individual to understand their work methods, needs, and issues encountered with the current version of the application. These interviews revealed issues of traceability and maintainability of products over the years. As the evolution changes were made directly on the product itself, the teams end up duplicating them to avoid completely overwriting the previous versions and to be able to search the differences over time.

Those scenarios are exemplified in Figure 2. The tables are a simplified extract of a grass field book. A field book is used by a field worker to gather all the required data. Those data are the main variability point, whether it be in time or space. The data are dependent on the species involved in the trial, but also dependent on the breeder or field worker<sup>2</sup> habits. A common breeder use case is to refine its observations. A criterion usually observed for a given scope (e.g., The yield of a plot) can be refined to a more precise scope (e.g., a subplot).

<sup>2</sup>The distinction between those roles varies according the company and its size.

The first variability-related functionality is to be able to create a template representing a model. This leads to scenario *Sc.1*.

**Sc.1** As a Breeder, I want to create a template named "grassesTemplate" that contains all the standard information required for a grasses trial which are "Species, Plot, Yield" to begin the trials promptly.

Then, when a template is derived to be used as a product, here for a specific trial, adaptations can be conducted at the product level, as represented by scenario *Sc.2*.

**Sc.2** As a field worker, I want to fill my trial field book about grasses following a template named "grassesTemplate" and adding the harvest date at filling time to support multiple harvest cycles.

In the two previous scenarios, the user uses a model definition as a template, but this is always a product definition, partial or not, that is actually done. The main problem lies in the capacity to seamlessly support sequences of cloning (templating, deriving), and adaptations on the model. As an extension of scenario *Sc.1*, it is thus crucial that what is perceived as templating by the end user can be done at any time from a final product, as every action refining the domain analysis is conducted at the product level. This leads to scenario *Sc.3*.

**Sc.3** As a Breeder, I want to create a new template based on the last grasses trial field book and add a piece of information about temperature collected on the plot thanks to IoT sensors to develop a species that can withstand high temperatures.

Additionally, it becomes naturally necessary to filter in and out information from the product to the template, keeping some information while removing some. This leads to scenario *Sc.4*.

**Sc.4** As a field worker, I want to retrieve the trial field book used during a previous campaign (without its "observations") to conduct a new one and take advantage of the latest improvements.

In the following, we will use these scenarios to illustrate and validate our proposed model.

## 2.3 State of the art

The tackled problem involves capturing changes over the product model by end-users, with any product at any step being a potential starting point for a clone, keeping all or part of it during the clone operation. In what follows, we will describe possible partial solutions proposed in the literature, focusing on clone-and-own, but also on operation-based modeling. We will also review research work related to spreadsheets as modeling and templating issues have been studied in this domain.

*Clone-and-own.* The clone-and-own approach [23, 33, 36, 46] involves copying existing artifacts from one product to another and modifying them as needed to fit the requirements of the new product. However, this approach can lead to unintentional divergence [33] between the products over time, resulting in model drift [53]. Model drift occurs when the models used in different products become increasingly different from each other, making it difficult to maintain consistency and quality across the product

line. This can result in increased efforts for tasks such as change propagation, domain analysis, and quality assurance.

Clone-and-own is an *ad-hoc* approach that should be *a priori* avoided, and many proposals have tackled different challenges related to migrating from this approach to the full SPL paradigm [4]. The main issue is to locate features [4, 19, 41] across the available artifacts and cloned variants [54], with the necessity of knowing the set of features in advance and a certain lack of precision in output [26, 28, 31, 39, 42, 43, 55]. However, clone-and-own is known to be mainly used in the industry [6], and several situations demonstrate that migrating to a full SPL with a platform managing features and mapping is risky and too costly [35] (e.g., when the number of products is low [45], or when the number and evolution of products is not known [23]), with possible loss of flexibility at the end [36].

Antkiewicz et al. have proposed a classification over a virtual platform to organize different progressive levels in the adoption process toward a full SPL [3]. In our context, cloning and adapting is a mechanism directly available to end-users, and a core part of their profession in building and evolving data-oriented processes, as in the agronomy domain. We thus believe that the exposed problem should be tackled by adding management support to the clone-and-own process, leading to a solution of type "L1: Clone-and-Own with Provenance" in Antkiewicz et al.'s classification [3], but with more control and automation in what can be done with the provenance information between clones.

*Managing clone-and-own product lines.* Several approaches supporting a form of variability management in clone-and-own product lines have been proposed [27, 33]. The basic idea is to utilize feature traces in a bottom-up manner for change propagation and the composition of new products [27]. Some approaches rely on the organization of a version control system to identify features and variants and detect inconsistencies between them, switching from code-based to feature-based reasoning [46, 47]. While the need for variant identification is important in our context, features are not actually present and cannot drive the representation of differences between clones. A recent experience report in the railway domain [53] shows that the model divergence is important in a model-driven SPL, and that differencing large models between a product and the platform, or between products is key. The authors then reuse semantic lifting of model differences [32] fed with high-level change patterns derived from model repositories [51, 52], so that a set of relevant differences can be used, and for example, filtered for change propagation. There are several similarities in our context. We are both model-driven and we also aim at taming the complexity of the model differences and the model drifts that occur between cloned or evolving products. However, as the end-users<sup>3</sup> are directly changing the model through high-level relevant changes, our problem is more to represent and trace completely the model differences than to mine them from other artifacts. For that purpose, we specifically study relevant work on modeling in the next paragraph.

Several authors also define variation control systems to unify variability management (features, variants, and variation points) and version control [38]. These systems aim at supporting the evolution and maintenance of software systems with many variants

<sup>3</sup>and developers editing starting examples in the main application.

Most of the time they provide support for commits of all variability information, but are completely feature-centric even when tackling a clone-and-own context [27]. Some recent advances [33] push further the integration of version control systems with SPL concepts, with the aim to automatically synchronize variants with change propagation based on a transparent collection of feature mappings.

Besides, a joint effort was also recently made to propose a unified model to represent variability in both space and time [2]. The conceptual model allows for representing features, feature revisions, as well as mapping to fragments for composition, but it is focused on a full SPL architecture with a platform, not on a clone-and-own context to be managed.

*Operation-based modeling.* Handling model differences [34, 37] is key in Model-Driven Engineering for comparing [50], versioning [1], or transforming models [7]. The work of Blanc et al. [8] defines operation-based modeling in which representing models involves breaking them down into sequences of elementary construction operations (create, add, setProperty, setReference...) [40]. This allows for a more uniform way of detecting and resolving structural inconsistencies between related models, regardless of their meta-model. This approach has been exploited in the SPL field to identify features in the source code of product variants [42, 55] or in UML models [5], but always with the aim of migration towards a common platform in a fully-fledged SPL. In our case, we aim at representing the differences between the data processing model and operation-based modeling seems to fit well. However, strictly applying this approach would lead to representing the differences at the finest grain with low-level operations on the model. This would create very large traces with no salient changes from a user perspective, and the need to retrieve them, as in proposals using semantic lifting [32] or detecting model drift [53] or domain-specific edits [52], as discussed in the previous paragraph. However, the approach could be adopted by representing the user-level operations on a data processing model so that all adaptation traces would be captured while being relevant to users. It must be noted that this would only partially solve the problem as variability-related operations creating or using a template are not directly supported by this kind of approach.

Besides, delta modeling [13, 49] is a modular way to capture variability by explicitly specifying the changes between system variants. A change is defined by so-called delta modules that specify modifications (e.g., add/remove a superclass, add/remove an interface) to be operated on a core part. Delta modules can then be composed to build products. Although the aim of difference capture is similar, in our case, our main focus is capturing successive model changes in order to trace all adaptations and clones.

*Spreadsheet technology.* Using spreadsheets is one of the easiest and the most flexible ways for many end-users to manipulate tabular data [10, 11, 48] with direct live coding and deployment [29]. While their expressiveness is seen as low [14] and large data sets do not scale well [15], research advances in the field of spreadsheets have notably focused on smell detection in formula [29], but also on extracting structural components and groups to reveal the underlying semantics [12, 20–22]. In our case, we are interested in the internal model of software that handle tabular data while providing more expressive ways to structure the dependencies between data

and processes, and thus, to better define the semantics. This can also be observed in all approaches building on model-driven engineering to describe a model of the spreadsheet (e.g., ClassSheet [24]), and mainly to reason from a spreadsheet on an inferred model [16–18] containing functional dependencies and formulas. Interestingly, to make a bidirectional transformation [18], Cunha et al. use a representation based on atomic operations [8] 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.

In conclusion, to the best of our knowledge, while different research results and approaches have tackled and sometimes solved a part of the defined requirements, no solution currently exists to meet all of them.

### 3 CONTRIBUTIONS

As discussed in the previous sections, we advocate that facing the cloning and adaptations that must be done by end-users on their data model while it is used, their model representation must be captured along with each action. Representing actions at a high level with an operation-based approach [8] should enable us to build a complete trace, thereby eliminating the unintentional and uncontrolled divergence in the model [53].

The model itself, which is presented in Section 3.1, will capture both the business of data handling and processes, with concepts of *Observation*, *Requirement*, and *Value*, and the trace of actions tracking the product evolution over time. On the business side, a product is made of observations on subjects, associated with a value, and constrained by explicit requirements on observations (which allows for representing in the model dependencies between observations, and consistency requirements between these dependencies). As for the tracing part, the product evolution is explicitly represented through linked trace objects that contain the different actions over the model (cf. Section 3.3). To support the four variability-related scenarios (cf. Sc.1 to Sc.4 in Section 2.2), specific variability-aware actions are captured in traces, creating explicit support for cloning and adaptation actions (cf. Section 3.4).

It must be noted that we currently restrain the modeling part to these observations, with no support for formulas that would compute values, as in a spreadsheet. It would be also interesting to study to which extent the business part of the model could be generalized or applied in other domains. These two points are part of our future work.

#### 3.1 Model

*3.1.1 Observation.* In our model, the central element is observation. It enables the connection between a criterion evaluated, an associated value, and the subjects being observed. Those concepts are the key components of the domain (cf. Section 2.1). The novelty and relevance of this model lie in its capacity to capture data dependencies and to enable reasoning based on the usage of data.

As for our grasses trial example, we consider we are observing the yield of two species (*Andropogon Gerardii* and *Andropogon Halli*) growing in a specific plot and harvested at a specific date. The terms "Species", "Plot", "Yield" and "Harvest date" constitute the *criteria* of the trial. A noted yield is about one of the species, in a plot at a

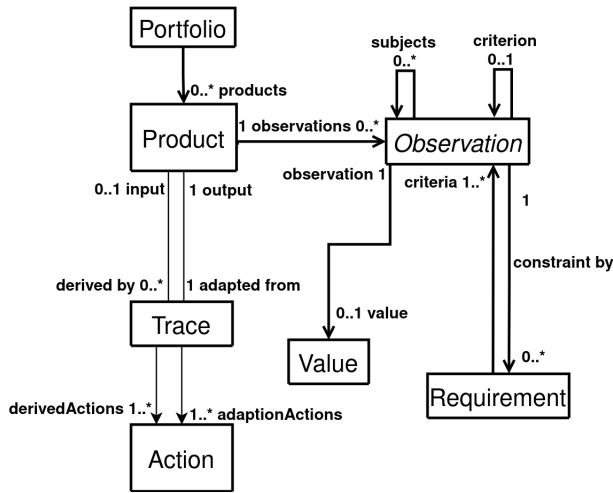


Figure 3: The proposed model

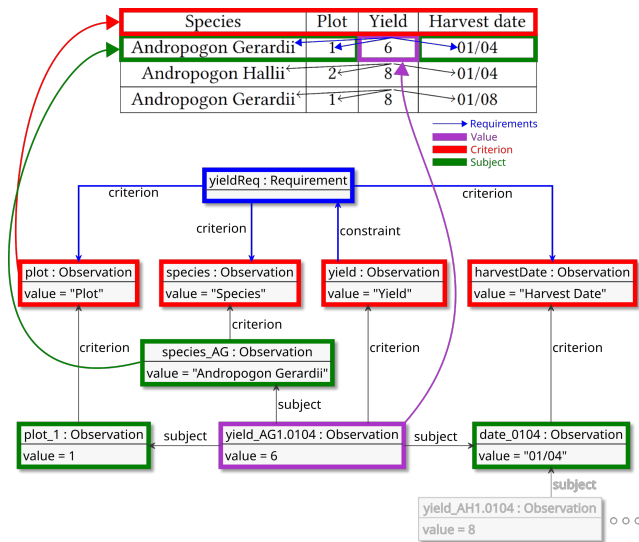


Figure 4: Partial instantiation of the model representing a field book

precise date, the three latter being the subjects of the observation evaluating the criterion "Yield".

The Figure 4 represents the tabular organization of a field book in our example after applying scenarios 1 and 2, as in Figure 2. A partial instantiation of the right part of our model (cf. Figure 3) on this field book represented and mapped to the field book. The bottom right grayed part corresponds to the rest of the instantiation not represented.

It must be noted that the three concepts (observation, criterion, subject) are by themselves observations. Notwithstanding its apparent peculiarity, the observation postulates that a user's data input can assume the roles of a subject, a criterion, or an observation in a way analogous to a cell in a spreadsheet. The distinction between the three types of entities comes from the usage made

of the information and the way it impacts the gathering and the post-process.

Our model can be mapped to the field book as follows:

- the first line corresponds to the criteria in the model which are traced in Section 3.2 under the instance of *species*, *plot*, *yield*, and *hd*
- all the cells in the *Yield* column are neither a criterion nor a subject but are the observations evaluating the criterion *yield* and dependent to subjects evaluating the criteria *species*, *plot* and *yield* as illustrated thanks to the arrows.
- all the other cells are subjects (observations evaluating the criteria *Species*, *Plot*, and *Harvest date*).

An **observation**  $o$  is defined by:

- a *value*  $v_o$  which is a reference to a datum;
- the *criterion*  $c_o$  it evaluates, if  $c_o$  is null,  $o$  is itself a criterion;
- the set of observed *subjects*  $s_o$ , if the set is empty and  $o$  evaluates a *criterion*,  $o$  is itself a subject;
- the *requirement*  $r_o$  associated with a criterion  $c_o$  enables the conformity check that all observations  $o_i$  evaluating  $c_o$  conform to the expectations noted  $conforms(r_o, o_i)$ .

We note  $o = (c_o, s_o, r_o, v_o)$ . Eponymous functions are defined, such as  $subjects(o) = s_o$ . We note the absence of value by *null* and any value by  $_$ .

For the sake of simplicity in the examples, we note the values directly in the tuple and the requirement only as a set of criteria to which the observations should refer.

The observation  $yield_{AG01.0104}$  in Figure 4 can then be represented as:  $o_{yag1.0104} = (yield, (ag, 1, 01/04), null, 6)$ . It is worth noting that  $r_o$  is, here, null since the requirement object is an aggregation of criteria defining the constraints applied to the criterion itself which is, then, transposed into the subjects' dependency on the observations evaluating the criterion.

A concept that can be considered unique within the business domain is represented by the same instance within our model when it's referenced within multiple contexts. For example, plot number 1 in the field book is considered unique and, therefore, is represented by a unique instance. The yields valued by 8 are, unlikely, specific to a set of subjects and are represented by distinct instances.

**Criterion characterization.** A **criterion** is an observation  $c$  that neither evaluates any other criterion nor observes any subject. It represents one of the aspects that should drive the observation by the person in charge of the trial (e.g., "Number of Tomatoes on the plant", "Maze's height", "Strawberry's redness (0 to 5)").

$$i.e., isCriterion(c) \Leftrightarrow criterion(c) = null \wedge subjects(c) = \{\}$$

**Subject characterization.** A **subject** is an observation  $s$  that evaluates one criterion and does not observe a subject. This characteristic offers the possibility to compose an observed object while keeping its components observable by themselves and correlate two observations about a common subject (e.g., making possible to correlate a species performance within a plot and the soil analysis of the plot).

$$i.e., isSubject(s) \Leftrightarrow criterion(s) \neq null \wedge subjects(s) = \{\}$$

**Observation properties.** An **observation**  $o$  is **well-formed** if it respects the following properties,  $isWellFormed(o) \Leftrightarrow$

- (1) it is a criterion, or it evaluates one criterion  
 $isCriterion(o) \vee (c = criterion(o), c \neq null, isCriterion(c))$
- (2) it only observes subjects, i.e.,  
 $\forall s \in subjects(o), isSubject(s)$
- (3) a requirement  $r$  references only criteria, i.e.,  
 $\forall o \in criteria(r), isCriterion(o)$
- (4) only criteria references requirements, i.e.,  
 $requirement(o) \neq null \Rightarrow isCriterion(o)$
- (5) it complies with the requirements of its criterion i.e.,  
 $\neg isCriterion(o) \Rightarrow$   
 $\forall c \in criterion(o), conforms(requirement(c), o)$

3.1.2 *Product*. As explained in Section 1 a product aggregates a set of observations and their relations. A **product**  $p$  is 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).

We note  $p = (n_p, O_p, t_p)$ . Eponymous functions are defined, such as  $observations(p) = O_p$  and derivate functions such  $criteria(p) = \{o \in O_p \mid isCriterion(o)\}$

*Product properties*. A **product**  $p$  is **well-formed** if it respects the following properties,  $isWellFormed(p) \Leftrightarrow$

- (1) All the observations are well-formed, i.e.,  
 $\forall o \in observations(p), isWellFormed(o)$ ,
- (2) The product is complete, i.e.,  
 $observations(p) = O_p, \forall o \in O_p$   
 $criteria(o) \in O_p$ ,  
 $subjects(o) \subseteq O_p$ ,  
 $criteria(requirement(o)) \subseteq O_p$
- (3) All the observations on the same criterion relate to different sets of subjects<sup>4</sup> keeping those observations distinguishable i.e.,  
 $\forall c \in criterion(p), \forall o_i \in observations(c)$ ,  
 $\neg \exists o_j \neq o_i, subjects(o_i) \cong subjects(o_j)$
- (4) Any observation respects the requirements of its criterion, i.e.,  
 $\forall c \in criterion(p), r_c = requirement(c) \neq null$ ,  
 $\forall o_i \in observations(c), conforms(r_c, o_i)$

We define the product  $p1$  corresponding to the table shown in Figure 4 as being formed by the set of observations defined previously:

i.e.,  $observations(p1) = \{species, \dots, hd1, y1\}$

A portfolio  $pf$  is a set of Products and is **well-formed** if all of its products are well-formed i.e.,  $\forall product(p) \in pf, isWellFormed(p)$  and all are named distinctly.

It is possible to find a product from its name. As we only work with one portfolio, we simply note:  $getProduct(productName) \rightarrow product$

## 3.2 Computed trace

A product conceived to be reused as a clone candidate (which can also be referred to as a template) is illustrated within Figure 2 by the

<sup>4</sup>If several observations are on the same criterion and the same subjects, then it is the same, or a subject is missing, which makes it possible to distinguish them, such as a date or an author.

first product issued from the empty set (scenario Sc.1). It contains the expected criteria and also the associated *Requirements* as expressed in the upper part of Figure 4. A cloned product contains the trace of all the actions required to reach its state and the references to the traces leading to its children.

The product trace is processable by the cloning actions to create a new product containing those criteria with their requirements and enables the adaptations without altering the source product (scenario Sc.2). The adaptations can relate to the data gathered (the subjects and the related observations) but also to the criteria they refer to. This is illustrated within the Figure 2 by the adaptations leading to the addition of the subjects *Andropogon Gerardii*, *Andropogon Hallii*, and their related data but also with the addition of the new criterion *Harvest date*. That additional information is part of the product but exists also through the trace of the products. Furthermore, as the cloning actions can be made from a product under use, scenario Sc.3 will be covered while filtering capabilities added to the cloning action (Subsequently, this filtered cloning action is referred to as "extraction action") will support scenario Sc.4. This latter scenario is depicted in Figure 2 by the conservation of the yield's observation related to the species *Andropogon Gerardii* in *plot number 1* and harvested the *01/04*. This filtered clone can be, then, adapted to collect a new additional set of data enabling business operations that are not in the scope of this paper (see Section 6).

## 3.3 Adaptation actions

We are defining the following independent and atomic actions<sup>5</sup>. Those actions can lead to inconsistent objects<sup>6</sup> that are, in this case, unaltered:

- $createProduct(name) \rightarrow Product$  : Create a new named product
  - Pre-conditions:
    - (1) there is no product with the same name in the current portfolio
  - Post-conditions:
    - (1) a well-formed named product exists in the portfolio and can be derived or adapted. It does not contain any observations.
- $createObservation(criterion, subjectList) \rightarrow Observation$  : Create a new observation
  - Pre-conditions:
    - (1)  $criterion$  can be null or must be an existing criterion
    - (2)  $subjectList$  can be empty or every element must be an existing subject
    - (3) the construction of the observation conforms with the criterion associated with it
  - Post-conditions:
    - (1) the new observation exists and is well-formed
- $valueateObservation(observation, value) \rightarrow void$  : Associate a value to a pre-existing observation
  - Pre-conditions:

<sup>5</sup>Observation removal are not supported as filtering is provided with the cloning actions (Section 3.4). The modification of a value (rather than the removal of an observation) is not supported in this context, and is closely related to the future work on formulas.

<sup>6</sup>Many inconsistencies may occur in traces. Detecting them is part of our future work, and some trails are discussed in Section 4.2.



- (1) *observation* exists and is well-formed
- Post-conditions:
  - (1) the observation exists, is valued, and is well-formed
- *addObservation(product, observation) → void* : Add an observation to a product
  - Pre-conditions:
    - (1) the product and the observation exist and are well-formed
    - (2) the observation's criteria exist and its requirements are respected
  - Post-conditions:
    - (1) the product exists and is well-formed
- *createRequirement(criterion, criteriaList) → Requirement* : Create a requirement for a specific *criterion*
  - Pre-conditions:
    - (1) the criterion exists, is valid, and is not associated with another requirement
    - (2) the criteria within the list exist and are well-formed
    - (3) all the criteria exist within the same product
  - Post-conditions:
    - (1) the product is well-formed (all observations associated with the criterion respect the requirement)

By design, the observations and the products are always well-formed.

*Example:*

- (1) *createProduct('GrassesTemplate') → p1*
- (2) *createObservation(null, {}) → species*
- (3) *valueateObservation(species, "Species")*
- (4) *addObservation(p1, species)*
- (5) ...
- (6) *createObservation(plot, {}) → p11*
- (7) *valueateObservation(p11, 1), ...*

**Trace 1: The trace leading to the product which can be mapped to the table in 4:**

At this stage, there are no requirements. According to the product *p1*'s characteristics, all the observations on the criterion *yield* are related to a species, a plot, and a harvest date. We can then create a requirement relating to the criterion *yield* traced as: *createRequirement(yield, {plot, species, hd})*.

This requirement creation inducts a conformity check that every observation on the criterion *yield* conforms to it and references a set of observations valuating the criteria defined by the requirement.

A trace  $t = [a_1, \dots, a_n]$  is **well-formed** if all the actions lead to a well-formed product.

Various filtering actions that do not modify the cloned products can be defined. We will not present them here since those actions are product and use case specific, although they are very useful.

### 3.4 Cloning actions

Based on the adaptation actions, we are now interested in actions that allow the cloning of existing products and extracting new products by limiting the embedded observations.

Those actions are only applicable to well-formed elements.

#### 3.4.1 Derivation.

*derive(originProduct, newProductName) → Product*

The derivation produces a new product which is a deep copy of the original product thanks to its trace. By design, this new product is well-formed. This action is advantageous in situations where it is required to adapt a product with additional data without altering the original version. The trace embedded within the new product is, therefore, a duplicate of the trace of the product which it is derived from. The derivation action is marked within the source product to trace its derivations. The derivation action generates atomic actions which are denoted as derived actions (da) (*derivedActions* in the model).

*Example:*

- (1) *derive(grassesTemplate, "grassTrialOne") → p2*
- (2) *da : createObservation(null, {}) → harvestdate<sub>p2</sub>*
- (3) *da : valueateObservation(harvestdate<sub>p2</sub>, "Harvestdate")*
- (4) *da : addObservation(p2, harvestdate<sub>p2</sub>)*

**Trace 2: The trace leading to the product p2 derivative of the GrassesTemplate product (first step of Scenario 2 in Figure 2).**

#### 3.4.2 Extraction.

*extract(originProduct, observationsList, newProductName) → Product x Listoflostobservations*

The extraction produces a new product in which only the requested observations have been cloned and adapted, only, that they allow for the construction of a well-formed product.

Observations that could not be cloned are returned as "lost". Observations are lost if the criterion associated with them is not part of the required observations. Subjects that are no longer among the requested observations will not be referenced by cloned observations. Observations can thus become subjects according to our definition.

In the implementation, this action first returns an atomic action list and the list of lost observations. The user can, then, carry out the extraction action by requesting the execution of the actions or reformulating the request.

*Example:*

- (1) *extract(p2, {species<sub>p2</sub>, plot<sub>p2</sub>, yield<sub>p2</sub>, harvestdate<sub>p2</sub>}, "trialTwo") → p3*
- (2) *da : createObservation(null, {}) → temp<sub>p3</sub>*
- (3) *da : valueateObservation(temp<sub>p3</sub>, "Temp")*
- (4) *da : addObservation(p3, temp<sub>p3</sub>)*

**Trace 3: The trace leading to the product p3 extracted from the p2 product (first step of Scenario 3 in Figure 2).**

The detailed process of a filtered extraction is as follows:

- (1) Retrieve the source product's trace  $a_1, \dots, a_n$

- (2) Remove every creation trace corresponding to an observation that is not kept:
- An observation about a non-kept criterion (*i.e.*, a criterion absent from the list of kept observations).
  - An observation about a non-kept subject (*i.e.*, a subject absent from the list of kept observations).
  - Every value of a non-kept observation

The trace leading to the execution of Scenario 4 is not presented here but can be obtained from our reproduction package (*cf.* Section 4.2).

## 4 EVALUATION

### 4.1 Measurements in an industrial case study

Doriane, our industrial partner, provides its customer with a tailored complex software system whose configuration can be refined by the end-user afterward. As some customer data were available for the usage of the application for several years (following the *current solution* organization of Figure 1), we conducted an experiment to validate our initial hypotheses and the relevance of our approach. One of the main objectives was to confirm that adaptations and cloning were actually present in the saved spreadsheet-oriented models. While we conducted interviews first (*cf.* Section 2.2), it appeared that the teams currently lack tools to compare products and trace their provenance. Nevertheless, all participants confirmed the derivation of products, particularly over multiple years to monitor experiment evolution and extract knowledge to start new trials.

We then extracted 6 portfolios from one of Doriane’s largest customers, each one representing a sub-domain of this customer. We gathered the products, and identified their usages thanks to the interviews. The extracted data are covering a period from 2004 to 2022 for a total of 242 413 trials carried out across all portfolios over 18 years. We then calculated the different input information that would be needed to build each element in our model. It must be noted that we did not couple this extraction tooling to our Prolog prototype (*cf.* next section) as it would need a generation of each action, which was not possible due to the lack of information about the dependencies among the data. Indeed the strength of our model should lie in its ability to track actions by modeling observations that take into account the observed subjects. However, automatically discovering these dependencies remains challenging in the general case and, in our experience, appears to depend on the organization of teams in its implementation. In the used portfolios, as everything was organized in columns, we could not automatically derive these dependencies, but we managed to obtain criteria and manually analyzed with application engineers their dependencies in data to determine subjects and observations. We extracted standard requirements such as the relationships between observations, dates, plants, and plots.

The resulting metrics are shown in Table 1. The number of “actual products” corresponds to the identified products at the beginning of 2023. However, this is not the real number of products, as the current version of the Doriane application cannot trace and name the cloned products. As we knew from the conducted interviews that a yearly release is quite common among customers and is at least a minimum, we estimated a number of “yearly-released products” with a yearly derivation decision point per product. We then calculate criteria,

Portfolio	# of actual products	# of yearly-released products (estim.)	# of criteria	# of subjects	# of observations	# of model actions (estim.)
A	16	288	989	272 835	11 137 621	11 411 445
B	32	576	1 328	518 783	40 210 197	40 730 308
C	29	522	1 224	99 690	5 841 945	5 942 859
D	21	378	1 046	47 170	2 962 399	3 010 615
E	18	324	2 781	9 637	3 254 960	3 267 378
F	11	198	1 297	148 133	5 513 299	5 662 729

**Table 1: Metrics of our model applied to our industrial case**

subjects, and observations. The subjects presented in Table 1 are only associated with the standard requirements described in the previous paragraph. Regarding model actions, we estimated their number by tracking atomic actions at the level of the tabular data, and divided the number by 3, as on average, 3 of these actions correspond to a high-level action in our model. This process is close, but simpler here, to the one used in semantic lifting for model differences [32, 52].

The different metrics on products, criteria, subjects, and especially observations, show that they are a huge number of adaptations, and cloning, in all portfolios over the years. On metrics themselves, we observe a large number of subjects in comparison with criteria, which correspond to the fact that the same criteria are used and reused across many different trials and thus subjects (*e.g.*, different plants, plots). This shows that adaptations and cloning, in the forms of the scenarios determined by our interviews, are present in all portfolios, demonstrating also their relevance. Overall these metrics also demonstrate that the underlying models at customer sites are really complex and of important size.

### 4.2 Prototype and reproduction package

To evaluate the capabilities of our approach to cover the scenarios defined in Section 2, we implemented as a proof of concept a prototype in Prolog. The main idea is to produce construction actions, as well as derive/extract actions, as Prolog facts, like in the work of Blanc et al. [8]. Thanks to the Prolog inference engine, it enables us to have a direct interpretation of the model based on predicate logic while maintaining each action as a first-class citizen. With this prototype, we are able to represent the model through Prolog facts and each Prolog definition is very close to the formal definition. For example, Listing 1 shows the Prolog facts that define the creation of a product, an observation, and a requirement, as well as the operation to add an observation to a product. The only additional statement used is the `trace` operation to force logging of the reasoning engine. The Prolog implementation of the whole action model, later with its extensions, will enable inconsistency detection [8] at many levels (*e.g.*, consistency of requirements with dependent observations, consistency of extracted product). Still, we believe that the separate formalization helps in providing a neat definition decoupled from Prolog implementation details.

```

1 createProduct(OutProductRef, ProductName) :-
2   gensym(ProductName, OutProductRef),
3   asserta(product(OutProductRef, ProductName)),
4   trace(createProduct, [OutProductRef, ProductName]).
5

```

```

6 createObservation(ProductRef,OutRef,Criteria, Subjects) :-
7   checkPrecondition(createObservation,createObservation(
8     ProductRef,OutRef,Criteria, Subjects)),
9   gensym(obs,OutRef),
10  asserta(obs(OutRef,Criteria,Subjects)),
11  checkRequirementOnCriteria(Criteria,OutRef),
12  trace(createObservation,[OutRef,Criteria, Subjects]).
13
14 addObservation(ProductRef,RefToObs) :-
15  checkPrecondition(addObservation,addObservation(ProductRef,
16    RefToObs)),!,
17  asserta(obs(ProductRef,RefToObs)),
18  trace(addObservation,[ProductRef,RefToObs]).
19
20 createRequirement(_ProductRef,RefToCriterion,RefToCriteria) :-
21  asserta(req(RefToCriterion,RefToCriteria)),
22  trace(createRequirement,[RefToCriterion,RefToCriteria]).

```

**Listing 1: Prolog facts representing the model**

In the implementation, the scenarios can be designed as an ordered sequence of operations also defined as Prolog facts. The four scenarios are available in the reproduction package as a single run of the Prolog engine. We added to the implementation an evaluator of the generated trace to show that it can be interpreted as defined in the model. To create a new product as a clone, our implementation then re-executes the trace of the targeted product.

A reproduction package is available online [9]. The Prolog code of the implemented model and the four scenarios are provided within a Dockerized environment containing SWI-Prolog to ease execution.

## 5 THREATS TO VALIDITY

On the application to the customer data provided by our partner, the first threat comes from the data treatment we applied. We had to determine by ourselves the derivation decision points within the extracted products extracted. This requirement came from the clone-and-own process currently implemented. To overcome this threat, we would have to iterate the whole generated traces with all the involved end-users at the customer site, as well as Doriane application engineers. Furthermore, it is obvious that a better validation could arise from a complete integration of the extracted facts from the customer data into our implementation. This is part of our plan for future work.

As for the Prolog prototype, the threats to validity are both internal, through the quality of its implementation, and external, with the scenarios coverage. The current implementation is only executed on the illustrative examples used in the paper, but we mitigated this threat by providing a trace generator that allows for re-interpreting the trace by the Prolog engine, leading to a kind of bootstrap of the tracing part. While the coverage of usage scenarios is by nature partial in comparison with the tackled problem, the fact that they were determined through interviews with our industrial partner gives us confidence in their relevance.

We also acknowledge two main limitations. First, the model and its traces are currently not exploited to reason on change propagation while it is a valuable feature to provide after making all differences explicit and complete. Still, we believe that our contribution is the first consistent step towards this goal. Second, we do not handle formulas as they certainly need specific handling to capture changes and reason about them. This is part of our future work.

## 6 CONCLUSION

Tabular data processing is at the center of almost all data-intensive applications. When tabular data are managed in dedicated software, more powerful than spreadsheets, they are mainly represented as column and row models. By the nature of the processes, such as research trials, these models are constantly cloned and adapted by the actions of the final users. Both the low-level nature of the models and the clone-and-own approach used then harm the ease of maintenance. - Our approach i) represents each high-level action as construction actions [8] over a model at the domain level so that adaptations are traced, and ii) represents extraction from a model to a template (actually another model) and derivation actions as the same kind of actions to trace cloning. As highlighted by four different scenarios, the generated trace keeps the whole history of model differences with explicit definitions of each clone, removing any model drift. A study conducted on customer data from our industrial partner shows the relevance of the supported scenarios to reason over all the changes and clones that happened over 19 years and could be properly captured. We have also validated the approach by implementing in Prolog the complete set of actions, as well as a trace generator that makes it possible to replay the generated traces on some small-scale examples.

In the short term, we plan to improve the experiments to validate the current state of the proposed model, bridging the Prolog prototype with the data extraction from customer data. This would need to obtain more information from the application engineers but should bring a more complete empirical validation and some insights on where to improve the model.

Two main research lines are part of our research plan in the longer term. First, we have to build on our model to define merging operations with inconsistency detection, so that change propagation support can be fully provided and experimented with. Second, we have to extend the model to also capture formulas or other source codes that are currently relating many elements in our models, just like in spreadsheets. To handle formulas, our operation-based approach could be too costly or simply not applicable as the edit actions could not be fully captured. It may be interesting to exploit the work done to extract models or semantic structures from spreadsheets [17, 22], as well as edit scripts generated from AST differencing [25] and semantic lifting from mined edits [52].

## 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] Sofia Ananieva, Sandra Greiner, Thomas Kühn, Jacob Krüger, Lukas Linsbauer, Sten Grüner, Timo Kehrer, Heiko Klare, Anne Koziolok, Henrik Lönn, et al. 2020. A conceptual model for unifying variability in space and time. In *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A-Volume A*. 1–12.
- [3] Michał Antkiewicz, Wenbin Ji, Thorsten Berger, Krzysztof Czarnecki, Thomas Schmorleiz, Ralf Lämmel, Ștefan Stănculescu, 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.
- [4] Wesley KG Assunção, Roberto E Lopez-Herrejon, Lukas Linsbauer, Silvia R Vergilio, and Alexander Egved. 2017. Reengineering legacy applications into software product lines: a systematic mapping. *Empirical Software Engineering* 22, 6 (2017), 2972–3016.
- [5] 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.

- [6] 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.
- [7] 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.
- [8] 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.
- [9] Nassim Bounouas, Mireille Blay-Fornarino, and Philippe Collet. 2023. *SPLC'23 Reproduction Package*. <https://doi.org/10.5281/zenodo.8111700>
- [10] 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.
- [11] Yolande E Chan and Veda C Storey. 1996. The use of spreadsheets in organizations: Determinants and consequences. *Information & Management* 31, 3 (1996), 119–134.
- [12] 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.
- [13] Dave Clarke, Michiel Helvensteijn, and Ina Schaefer. 2010. Abstract delta modeling. *ACM Sigplan Notices* 46, 2 (2010), 13–22.
- [14] Samuel Clemens. 2011. Five Ways To Tell You Have Outgrown Excel. <https://www.insightssquared.com/blog/5-ways-to-tell-you-have-outgrown-excel/>
- [15] 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/>
- [16] Jácome Cunha, Martin Erwig, Jorge Mendes, and João Saraiva. 2016. Model inference for spreadsheets. *Automated Software Engineering* 23 (2016), 361–392.
- [17] 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.
- [18] 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.
- [19] 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>
- [20] 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*.
- [21] 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.
- [22] 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.
- [23] 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.
- [24] 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.
- [25] Jean-Rémy Fallier, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus. 2014. Fine-grained and accurate source code differencing. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. 313–324.
- [26] 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.
- [27] 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.
- [28] 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.
- [29] Felienne Hermans, Bas Jansen, Sohoo 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.
- [30] 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.
- [31] 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.
- [32] Timo Kehrer, Udo Kelter, and Gabriele Taentzer. 2011. A rule-based approach to the semantic lifting of model differences in the context of model versioning. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. IEEE, 163–172.
- [33] 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.
- [34] 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.
- [35] 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.
- [36] 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.
- [37] 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.
- [38] Lukas Linsbauer, Thorsten Berger, and Paul Grünbacher. 2017. A classification of variation control systems. *ACM SIGPLAN Notices* 52, 12 (2017), 49–62.
- [39] 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.
- [40] Ernst Lippe and Norbert Van Oosterom. 1992. Operation-based merging. In *Proceedings of the fifth ACM SIGSOFT symposium on Software development environments*. 78–87.
- [41] 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.
- [42] 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.
- [43] 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.
- [44] Celina M Olszak and Ewa Ziemia. 2007. Approach to building and implementing business intelligence systems. *Interdisciplinary Journal of Information, Knowledge, and Management* 2, 1 (2007), 135–148.
- [45] Klaus Pohl, Günter Böckle, and Frank J van Der Linden. 2005. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer Science & Business Media.
- [46] 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.
- [47] 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.
- [48] 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.
- [49] Ina Schaefer, Lorenzo Bettini, Viviana Bono, Ferruccio Damiani, and Nico Tanzarella. 2010. Delta-oriented programming of software product lines. In *Software Product Lines: Going Beyond: 14th International Conference, SPLC 2010, Jeju Island, South Korea, September 13-17, 2010. Proceedings 14*. Springer, 77–91.
- [50] Matthew Stephan and James R Cordy. 2013. A Survey of Model Comparison Approaches and Applications. *Modelsward* (2013), 265–277.
- [51] 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.
- [52] 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.
- [53] 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.
- [54] 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.
- [55] 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.