Automated Analysis of iPricings Technical Report

Version 1.0

Alejandro García-Fernández, José Antonio Parejo, Francisco Javier Cavero, Antonio Ruiz-Cortés

SCORE Lab, I3US Institute, Universidad de Sevilla, Seville, Spain

{agarcia29,japarejo,ptrinidad,aruiz}@us.es

July 22, 2024

INDEX

Chapter 1

Introduction

Software as a Service (SaaS) has grown rapidly in popularity in recent years [9], driven by its subscription-based model, which adapts to a wide range of user needs. Modern pricing models for SaaS [5] have evolved to include multiple interconnected elements: features, usage limits, plans, and add-ons; enabling a wide range of different configurations (a.k.a. subscriptions). Although this adaptability benefits both users —by providing flexibility— and providers —by boosting revenue—, it significantly increases the complexity of managing variability within these systems [7].

A major challenge stems from the explosive growth of the configuration space, i.e. the set of all potential subscriptions that can be derived from a pricing model. For instance, Salesforce's pricing in July 2024, which included three plans and 14 add-ons, allowed for up to 12,544 unique configurations.[1] Offering such an amount of configurations is error-prone and difficult to manage, especially when adapting to pricing evolution or handling dependencies across multiple services.

This challenge can be addressed by introducing intelligent pricings (iPricings) [3], which are dynamic, machine-readable pricing models designed for SaaS products. Unlike ad-hoc solutions that companies often develop to address specific needs, that may offer short-term functionality but lack scalability and consistency, iPricings provide a structured approach to pricing management. By treating pricings as programmable and automatically adaptable artifacts, iPricings reduce the need for human intervention in pricing-driven service operations, enhance scalability, and align pricing strategies with broader SaaS business objectives, enabling more effective and sustainable analysis and management.

Given this context, we propose leveraging Automated Analysis (AA) methods to address the challenges of managing and optimizing iPricings. In the broader field of information systems engineering, AA has proven highly effective in extracting valuable insights from complex models to support decision-making and configuration processes, as demonstrated by foundational research on feature models [1, 2]. Furthermore, these techniques have successfully been applied in service engineering to manage service variability and configurations [14]. However, their application to SaaS, particularly in the context of iPricings, remains largely unexplored, leaving configuration artifacts underutilized despite their critical role in formal modeling and analysis. Notably, initial efforts such as those in Pricing4SaaS [5, 6] have begun to address this gap by introducing preliminary specifications for SaaS pricings, underscoring the need for more comprehensive AA frameworks that can address the specific challenges of pricing-driven SaaS development and operation [4].

Building on these challenges and the potential of AA to address them, our main contributions are as follows:

[1] Salesforce July 2024 pricing can be found here.

- We propose a formalization of iPricings as Constraint Satisfaction Problems (CSPs) that explicitly capture pricing and subscription constraints and enable automated analysis operations to be performed.

- We define seven fundamental operations to uncover latent, non-trivial insights within pricings, aiding SaaS users in managing subscriptions more effectively and empowering pricing designers to make well-informed decisions.

- We validate these operations using a dataset of more than 150 real-world pricing configurations [7] and a dataset of synthetic pricing with seeded consistency errors [8], both of which are made available for use in future research.

- We provide a reference implementation of the proposed operations to facilitate future research and practical applications.[2]

The remainder of this report is structured as follows: Chapter 2 presents the core concepts, challenges, and research advancements in SaaS pricing. Next, Chapter 3 introduces an approach to automate iPricing analysis by formalizing them as CSPs. After that, Chapter 4 presents a reference implementation and validation results. Finally, Section 5 provides a summary and outlines directions for future research.

[2]A sample final product: https://sphere.score.us.es/pricings/sphere/Buffer?collectionName=CAISE%202025

Chapter 2

Background

2.1 Pricing

A pricing is a component of the customer agreement, structuring the *features* of a service —defined as the distinctive characteristics whose presence/absence may guide a user's decision towards a particular subscription [4]— into *plans* and *add-ons* to control users' access to such features. While users can only subscribe to one of the available plans, they can subscribe to as many add-ons as they want since they are available for the contracted plan. As can be noted, in this domain, a feature encompasses both *functional features*, which constitute the core software product, and *extra-functional features*, which, while external to the product itself, enhance its perceived value (e.g., support, SLA guarantees, etc). This broader definition contrasts with the approach used in Software Product Lines (SPLs) [1], where feature models focus mainly on functional features, aiming to represent all possible product variants. By including both types of features, pricings capture a more comprehensive view of what influences customer value, aligning the service offering with the overall customer agreement.

	BASIC FREE	PRO $15.99 per user/month	BUSINESS $21.99 per user/month
Max assistants per meeting	100	100	300
Max time per meeting	40 mins	30 hours	30 hours
Recordings cloud storage	-	5 GB	5 GB
Automated subtitles	✓	✓	✓
Reports		✓	✓
Voting in meetings		✓	✓
Phone Dialing		Add-On	Add-On
LTI integration			✓
Administrator portal			✓
End-to-end encryption	✓	✓	✓
Chat support		If more than $50 invoiced	

ADD-ONS

Huge meetings
$50 / month

Translated captions
$5 / month

Phone Dialing
$100 / month

Figure 2.1: Excerpt of Zoom's pricing with 13 features, three plans and three add-ons.

In order to illustrate the upcoming concepts, we will use a real-world running example: Zoom. This is a cloud-based video conferencing service that enables users to virtually meet with others and optionally record the sessions to watch them later. An excerpt of its pricing, consisting of 13 features, is presented in Fig. 2.1. As can be noted, ten features are managed by plans, and three by add-ons. The pricing also imposes usage limits on the "meetings" feature, such as "max participants per meeting" and "max time per meeting", meaning that although the feature is available in all plans, the extent of their usage differs —higher-priced plans offer higher limits.

2.2 iPricings Analysis

Despite growing interest, there has been limited progress in the modeling and analysis of iPricings. The Pricing4SaaS model (see Fig. 2.2), introduced as a formal metamodel, represents one of the first generalized attempts to standardize pricings [5].Through this model, pricings are defined as a combination of *plans* and *add-ons* that regulate access to a set of features whose usage may be limited by usage limits. Additionally, its YAML-based serialization, Pricing2Yaml 2.3, has served as the initial step towards automated tooling for pricing-driven dev-ops [6] and thus iPricings.

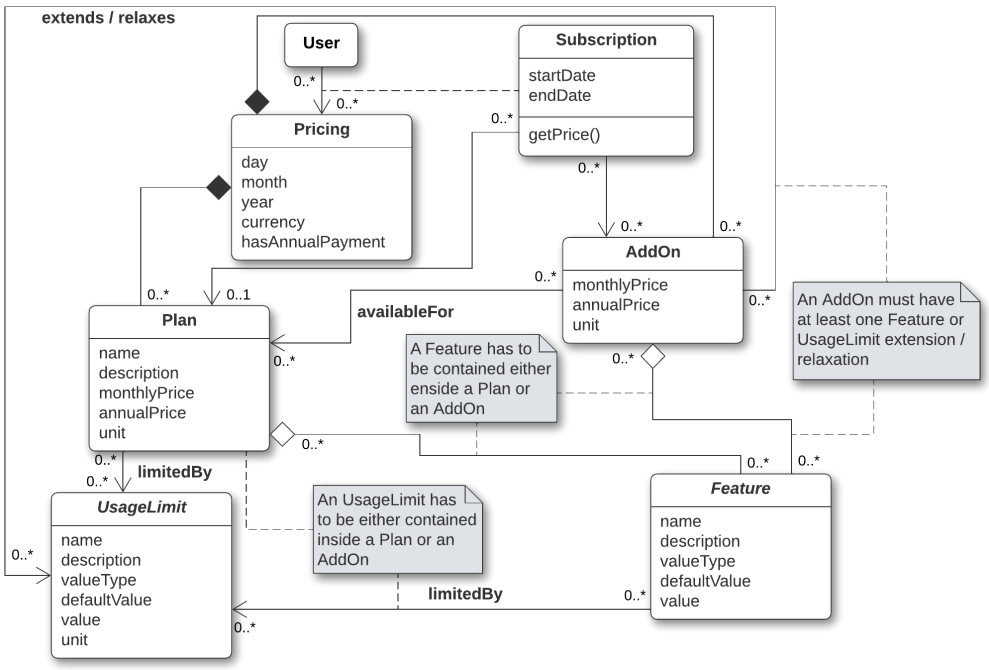

Figure 2.2: Excerpt of Pricing4SaaS UML Model. Created from the original in [5].

Building on this foundation, the study undertaken by [7] represents the first major effort to systematically analyze SaaS pricings. By modeling up to 162 pricings from 30 different SaaS providers –up to 6 different pricing versions between 2019 and 2024 per SaaS– this study not only demonstrated the flexibility of Pricing2Yaml (and, by extension, the adaptability of Pricing4SaaS) but also revealed significant trends in the evolution of pricings.

To perform their analysis and measure such evolution, the authors introduced two fundamental concepts about pricings:

- **Configuration Space:** the set of different subscriptions within a pricing.
- **Evaluation Space:** the minimum set of feature toggles that have to be evaluated in order to operate the functional features governed by a pricing, it enables DevOps teams to predict the impact of changes on pricing elements within the service.

Their results revealed an exponential growth of the configuration space, primarily due to the addition of add-ons over the last years, which diversifies available subscriptions and increases customization. Moreover, the study also highlights a widening gap between this configurability and the actual capabilities of feature toggling tools, which often struggle to manage the increasing variability. This disparity indicates that existing technical support for handling such challenges remains limited, underscoring a need for automated tools to reduce complexity and time-to-market while adapting to evolving customer demands.

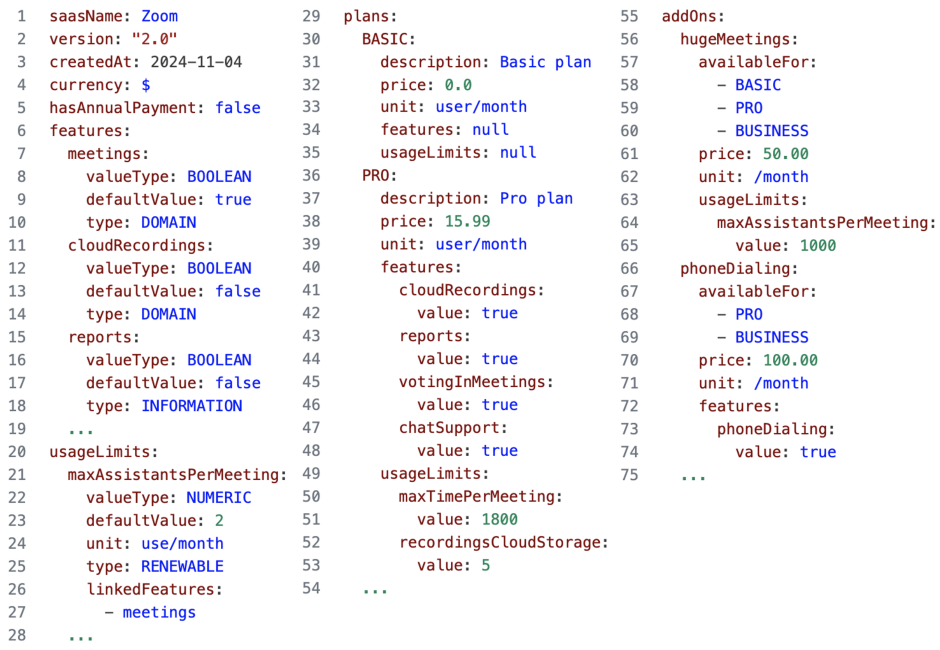

Figure 2.3: Zoom's pricing excerpt serialized in Pricing2Yaml

Chapter 3

Automated Analysis of iPricings

Given the gap between the rapid evolution of pricings complexity and the limited ability of feature toggling tools to manage their upcoming variability, automating the analysis of these structures is essential. Automation enables real-time validation of pricing specifications, ensuring structural consistency and providing critical insights during design and improvement phases. Our approach centers on the automated analysis of pricings serialized in Pricing2Yaml (see Fig. 2.3) since, to the best of our knowledge, it is the only serialization of pricings proposed in the literature. This will enable the detection of potential errors, such as inconsistent dependencies, and calculation of valuable metrics, like the cardinality of the configuration space —tasks that would be impractical to perform manually.

In what follows, we present our approach for the automated analysis of Pricing2Yaml specifications using constraint programming. In particular, we first present the formal semantics of Pricing2Yaml by explaining how these specifications can be mapped to a constraint satisfaction problem (CSP). Then, we present a catalog of seven analysis operations of Pricing2Yaml specifications and show how they can be automated using standard constraint programming reasoning operations.

3.1 Formal Semantics of Pricing4SaaS

The primary objective of formalizing Pricing4SaaS is to establish a sound basis for automating the analysis and decision-making processes in the context of SaaS pricing models, thus enacting the vision of iPricings. Following the formalization principles defined by [13], we follow a transformational style by translating Pricing2Yaml specifications to a target domain suitable for the automated analysis (*Primary Goal Principle*). Specifically, we propose Constraint Satisfaction Problems (CSPs) as this target domain, enabling pricings to be analyzed using constraint programming tools. This approach aligns with previous efforts to automate the analysis of feature models [2] and service level agreements [11, 10].

To define a CSP formally, it is expressed as a 3-tuple (V, D, C), where V is a set of variables, D represents the domains for these variables, and C is a set of constraints among the variables. A solution to a CSP is an assignment of values from each domain in D to the variables in V that satisfies all constraints in C.

Model-based CSP frameworks, such as MiniZinc[1], facilitate the formulation and resolution of CSPs by separating them into two distinct components [12]: (a) the model, which defines the structure of a class of problems; and (b) the data, which specifies a particular problem within this class. The pairing of a model with a particular dataset is a model instance (sometimes abbreviated to instance). This separation allows a single model to be applied across various

[1] https://docs.minizinc.dev/en/stable/index.html

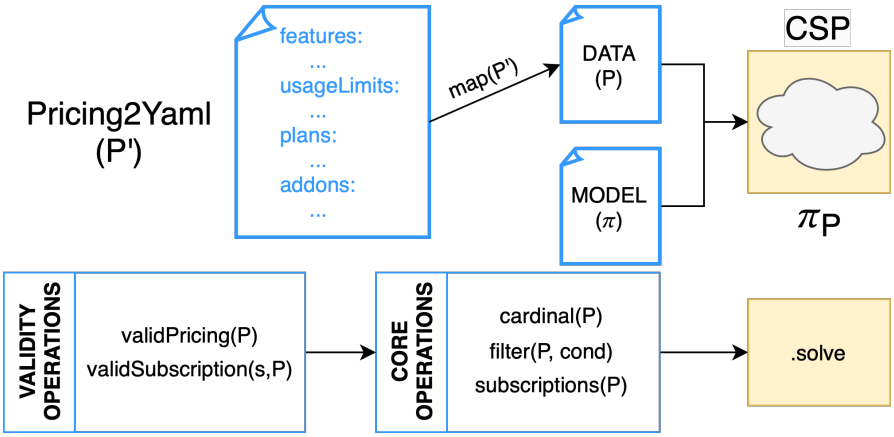

Figure 3.1: Outline of our approach to automate the analysis of Pricing2Yaml serializations

datasets, providing flexibility and scalability.

In our approach (see 3.1), a generic pricing model —with P_p plans and P_A add-ons— can be represented as a CSP model π that captures the following basic structural constraints derived from the semantics of pricings:

1. For each plan $p \in P_p$, if usage limits are defined, the features affected by those usage limits must be included within the same plan.

$$containsLinkedFeatures(p) \iff \begin{cases} p.usageLimits \neq \emptyset \implies \\ \forall u \in p.usageLimits, \\ u.linkedFeatures \subseteq p.features \end{cases}$$

2. If the pricing includes both plans and add-ons, each add-on $a \in P_A$ must be available for at least one plan.

$$isAvailable(a) \iff ((P_p \neq \emptyset \land P_A \neq \emptyset) \implies a.availableFor \neq \emptyset)$$

3. A subscription cannot be empty; it must include either a plan p with zero or more add-ons (the set of selected add-ons is denoted as A_S) if the pricing includes plans, or at least one add-on if the pricing consists solely of add-ons.

$$notEmpty(p, A_S) \iff (p \neq \emptyset \lor A_S \neq \emptyset)$$

4. If the pricing includes both plans and add-ons, and a subscription includes add-ons, every add-on must be available for the specific plan selected in that subscription.

$$addonsAvailable(p, A_S) \iff \begin{cases} p \neq \emptyset \land A_S \neq \emptyset \implies \\ \forall a \in A_S, \\ p \in a.availableFor \end{cases}$$

5. If a subscription includes add-ons with dependencies on other add-ons, all required dependencies must also be part of the subscription.

$$dependencyAware(A_S) \iff \begin{cases} A_S \neq \emptyset \implies \\ \forall a \in A_S, \forall a' \in a.dependsOn, \\ a' \in A_S \end{cases}$$

6. If a subscription includes add-ons that exclude others, all excluded add-ons cannot be included within the subscription.

$$exclusionAware(A_S) \iff \begin{cases} A_S \neq \emptyset \implies \\ \forall a \in A_S, \forall a' \in a.excludes, \\ a' \notin A_S \end{cases}$$

In this way, we can formally describe π as:

$$\pi \iff \left\{ \begin{array}{ll} \forall p \in P,\ containsLinkedFeatures(p) & (1) \\ \wedge\, \forall a \in P_A,\ availableFor(a) & (2) \\ \wedge\, \forall (p, A_S) \in P_p \cup \emptyset \times \mathbb{P}(P_A),\ notEmpty(p, A_S) & (3) \\ \wedge\, \forall (p, A_S) \in P_p \cup \emptyset \times \mathbb{P}(P_A),\ addonsAvailable(p, A_S) & (4) \\ \wedge\, \forall (p, A_S) \in P_p \cup \emptyset \times \mathbb{P}(P_A),\ dependencyAware(A_S) & (5) \\ \wedge\, \forall (p, A_S) \in P_p \cup \emptyset \times \mathbb{P}(P_A),\ exclusionAware(A_S) & (6) \end{array} \right\}$$

Then, we denote every model instance as π_P, created by pairing the model π with the particular data of a pricing P. Solutions to π_P thus represent the set of feasible configurations (a.k.a subscriptions), for a pricing P, where all defined constraints are satisfied. Formally:

$$solve(\pi_P) = \{(p, A_S) \in P_p \cup \emptyset \times \mathbb{P}(P_A) \mid \text{all constraints in } \pi \text{ are satisfied}\}$$

3.2 Analysis Operations

Building on the formalization outlined in the previous section, where iPricings were mapped to CSPs, we now turn our attention to the analytical operations that this formalization enables. These operations are key to extracting meaningful, latent insights from iPricing specifications. To provide clarity and continuity, each operation is illustrated using the Zoom's pricing excerpt (see Fig. 2.1) as a running example.

3.2.1 Number of Configurations

One of the most important metrics derived from a pricing is the cardinality of its configuration space, representing the number of different configurations that can be generated from it. A higher cardinality indicates increased flexibility for the SaaS, but also adds complexity to its maintenance.

Definition 1 (Cardinality)**.** *Let P be a pricing, the number of configurations of P, hereinafter cardinal, is equal to the number of solutions of its equivalent CSP π_P.*

$$cardinal(P) = |solve(\pi_P)|$$

In the example of Zoom's pricing excerpt, $cardinal(Zoom) = 20$. However, simply by adding a new plan, the cardinality increases to 28. In contrast, introducing an additional add-on doubles the cardinality, resulting in $cardinal(Zoom) = 40$. As demonstrated in [7], each new add-on leads to an exponential increase in cardinality, whereas adding a new plan results in only a linear increase.

3.2.2 Filter

An operation is needed to limit the number of potential configurations within the pricing. This is particularly useful for customers that are looking for a configuration with specific features and usage limits, i.e. they are not interested in all possible configurations but the ones that tailor to their needs (the ones that pass the filter).

Definition 2 (Filter)**.** *Let P be a pricing and F a constraint representing a filter, containing the desired features and usage limits' values within the subscription. The filtered pricing of π_P, hereinafter filter, is equal to π_P with the added constraint F.*

$$filter(P, F) \iff (\pi_P \wedge F)$$

A possible filter for zoom's pricing excerpt would be to ask for all configurations with the "administrator portal" feature and, at least, a 200 assistants capacity in meetings. By applying this filter to the pricing, the number of potential configurations decreases from 20 to 8.

$$F = (administratorPortal = true \wedge maxAssistantsPerMeeting >= 200)$$

$$cardinal(filter(Zoom, F)) = 8$$

3.2.3 Subscriptions

Once π_P is defined, there should be a way to get all the solutions of the CSP, i.e. possible configurations of the pricing.

Definition 3 (Subscriptions)**.** *Let P be a pricing, the potential configurations of the pricing, hereinafter subscriptions, are equal to the solutions of the equivalent CSP π_P.*

$$subscriptions(P) = \{s \in solve(\pi_P)\}$$

Within zoom, we would like to get all possible subscriptions that include the "records" feature. In this way, $P = filter(Zoom, records = true)$ and $subscriptions(P) = \{s \in solve(\pi_P \wedge records = true)\}$.

3.2.4 Subscription Cost

In order to perform cost optimization operations, it is necessary to define how the cost of a subscription is computed.

Definition 4 (Subscription Cost)**.** *Let s be a subscription of the pricing P. The cost associated with s is calculated as the sum of the selected plan's price, if it exists, and the prices of the selected add-ons, if any.*

$$cost(s) = \begin{cases} s.plan.price & s.addons = \emptyset \\ \sum_{a \in s.addons} a.price & s.plan = \emptyset \\ s.plan.price + \sum_{a \in s.addons} a.price & otherwise \end{cases}$$

Within the zoom's pricing excerpt, we would like to get the cost of a subscription that includes the plan "PRO", and the add-on "Huge Meetings", i.e., $s = (PRO, \{HugeMeetings\})$. Thus, $cost(s) = 65.99$.

3.2.5 Pricing Validation

A pricing is valid when it has at least one configuration that can be selected. That is, a model where π_P has at least one solution.

Definition 5 (Valid Pricing)**.** *A pricing P is valid if its equivalent CSP π_P is satisfiable.*

$$valid(P) \iff subscriptions(P) \neq \emptyset$$

As an illustration, let $CircularConstraints$ be a pricing that consists solely of three add-ons $P_A = \{a_1, a_2, a_3\}$, that share the following restrictions among them: i) a_1 depends on a_2, ii) a_2 depends on a_3, and iii) a_3 excludes a_1. Then:

$$valid(CircularConstraints) = false$$

3.2.6 Subscription Validation

Since many different subscriptions can be generated from a pricing, it should be possible to contrast whether a subscription is valid according to a pricing.

Definition 6 (Valid Subscription)**.** *Let P be a pricing and s be a subscription, that is considered valid if and only if it is a potential solution of π_P.*

$$valid(s, P) \iff s \in subscriptions(P)$$

Taking the excerpt of zoom's pricing as an illustration, a subscription that includes the usage limit "maxAssistantsPerMeeting = 1200" is not valid, as the maximum permitted value for this usage limit is 1000, achievable only by purchasing the "Huge Meetings" add-on.

3.2.7 Optimum Subscription

Identifying the optimum subscription based on a specific criterion is essential in pricing for both customers and providers. Customers can determine the minimum price for a defined set of features and usage limits, while providers gain insight into the price range at which a particular set of features and usage limits is offered.

Definition 7 (Optimum)**.** *Let P be pricing and O an objective function for the subscription cost, then the optimum set of subscriptions, hereinafter max and min, is equal to the optimum space of π_P.*

$$max(P, O) = max(\pi_P, O)$$
$$min(P, O) = min(\pi_P, O)$$

It is also possible to apply a filter to the zoom's pricing excerpt and then ask for an optimal subscription. Thus, a possible optimum criterion for our running example would be to ask for all subscriptions with $record = true \wedge cloudStorage >= 5$, and the minimum value for the subscription cost. In this case, optimum subscriptions S_{opt} are:

$$\begin{cases} P & = filter(Zoom, record = true \wedge cloudStorage >= 5) \\ O & = cost \\ S_{opt} & = min(P, O) \end{cases}$$

Chapter 4

Tooling Support and Validation

This section presents a practical implementation of the proposed CSP-based approach to perform AA on iPricings using MiniZinc [12], detailing its structure and functionality. It also validates the solution using real-world and synthetic datasets to evaluate its effectiveness and identify potential limitations.

4.1 From Web pricings to iPricings

To bridge the gap between web pricings and iPricings, we adopted the Pricing4SaaS model [5]. A key factor in selecting this approach was its YAML-based serialization, Pricing2Yaml, whose expressiveness has already been validated in prior studies [7]. It enables the transformation of web pricings into operational iPricings, providing a structured and actionable representation of them.

In addition, to further support the creation and management of iPricings using the Pricing2Yaml syntax, and by extension the Pricing4SaaS metamodel, we have developed an online editor for this syntax that has been integrated into SPHERE, a platform for intelligent pricing-driven solutions.[1] While these tools and methodologies simplify the transition from static web pricings to iPricings, they are part of an ongoing effort to progressively enhance the management of pricing-driven development and operations, incorporating continuous improvements to address emerging challenges.

4.2 iPricings formalization in CSP: MiniZinc

MiniZinc is a medium-level declarative modeling language widely recognized for its flexibility and compatibility with various constraint programming (CP) solvers. By separating CSPs into two components —the model, which defines the constraints and structure of a class of problems, and the data, which specifies the parameters for a particular instance— MiniZinc allows a single model to be reused across multiple datasets, enhancing both scalability and modularity (see 2). Combined with its ability to translate these problems into the FlatZinc format, MiniZinc ensures compatibility with a wide range of solvers, effectively bridging the gap between high-level problem descriptions and the low-level representations required by solvers [12].

Building on MiniZinc's ability to separate models and data, we have structured our iPricing formalization using a modular and hierarchical approach. The set of models we developed, as shown in Fig. 4.1, reflects a bottom-up methodology. Starting with a base model (**Pricing-Model.mzn**), we define the fundamental parameters and variables that underpin the problem. The parameters, derived from the input data, capture the specifics of the pricing plan, while the

[1] The editor is accessible at: SPHERE Pricing2Yaml Editor

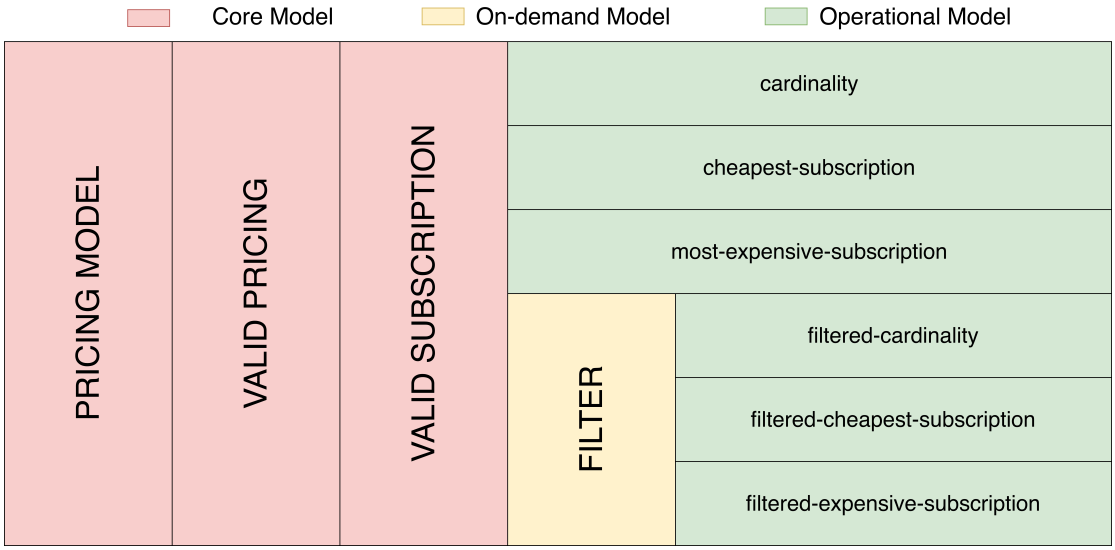

Figure 4.1: Structure of MiniZinc models to represent the formalization of iPricings

variables represent the attributes of subscriptions that form the solutions of the CSP. This foundation enables us to incrementally add layers of constraints and operations tailored to different analytical tasks.

Expanding upon *PricingModel.mzn*, we defined two models that introduce some extra constraints: i) **valid-pricing.mzn**, to ensure the consistency of the pricing received through the data file; and, over it, ii) **valid-subscription.mzn**, which assures the validity of subscriptions generated as CSP solutions. Using the latter as a base, we implemented a first set of analysis operations:

1. Cardinality (**configuration-space.mzn**): when executed to find all solutions, computes the entire configuration space of valid subscriptions.

2. Cheapest Subscription (**cheapest-subscription.mzn**): Implements a "min" operation with the objective of minimizing subscription cost.

3. Most Expensive Subscription (**most-expensive-subscription.mzn**): Performs a "max" operation with the objective of maximizing subscription cost.

Additionally, we defined over the *valid-subscription* model a filtering operation (**filter.mzn**), that allows defining specific criteria —such as a cost range (minimum and maximum thresholds), a required set of features and/or constraints on usage limits— and applying them to restrict the CSP's solution space to only those subscriptions that meet these conditions. Building on this filter, we developed equivalent versions of the previously mentioned operations, adapted to consider exclusively the filtered subset of valid subscriptions (**filtered-configuration-space.mzn**, **cheapest-filtered-subscription.mzn**, **most-expensive-filtered-subscription.mzn**).

To execute the operations described, the framework relies on a data file in *.dzn* format, which specifies the necessary parameters' values to construct the final CSP, according to the approach of MiniZinc. These data files are generated through a custom-developed converter that we implemented as part of the Pricing4TS library —a cutting-edge library, part of the Pricing4SaaS suite, for Pricing-driven DevOps [7]. This converter translates models defined in Pricing2Yaml into the .dzn format required by MiniZinc.[2] Tab. 4.1 provides a detailed definition of all the attributes included in *.dzn* files generated from an iPricing, along with the domain of their values, offering documentation of the structure of these datafiles. In addition, and as an illustration, Listing 4.1 presents the MiniZinc data file derived from Zoom's excerpt iPricing (Fig. 2.1).

[2]The converter is available at: https://github.com/Alex-GF/Pricing4TS/tree/main/src/utils/dzn-exporter.

Attribute	Type	Domain	Matrix Dimensions
num_features	int	$\mathbb{N}_0$	-
num_usage_limits	int	$\mathbb{N}_0$	-
num_plans	int	$\mathbb{N}_0$	-
num_addons	int	$\mathbb{N}_0$	-
features	array	string	-
usage_limits	array	string	-
plans	array	string	-
addons	array	string	-
plans_prices	array	$\mathbb{N}_0$	-
addons_prices	array	$\mathbb{N}_0$	-
boolean_usage_limits	array	$\{0,1\}$	-
linked_features	array2d	$\{0,1\}$	$\|USAGE\ LIMITS\|$x$\|FEATURES\|$
plans_features	array2d	$\{0,1\}$	$\|PLANS\|$x$\|FEATURES\|$
plans_usage_limits	array2d	$\mathbb{N}_0$	$\|PLANS\|$x$\|USAGE\ LIMITS\|$
addons_features	array2d	$\{0,1\}$	$\|ADDONS\|$x$\|FEATURES\|$
addons_usage_limits	array2d	$\mathbb{N}_0$	$\|ADDONS\|$x$\|USAGE\ LIMITS\|$
addons_usage_limits_extensions	array2d	$\mathbb{N}_0$	$\|ADDONS\|$x$\|USAGE\ LIMITS\|$
addons_available_for	array2d	$\{0,1\}$	$\|ADDONS\|$x$\|PLANS\|$
addons_depends_on	array2d	$\{0,1\}$	$\|ADDONS\|$x$\|ADDONS\|$

Table 4.1: Attributes of a Minizinc iPricing data file

Listing 4.1: MiniZinc datafile of Zoom's pricing excerpt

```
num_features = 11;
num_usage_limits = 3;
num_plans = 3;
num_addons = 3;

features = ["meetings", "cloudRecordings", "automatedSubtitles",
            "reports", "votingInMeetings", "phoneDialing",
            "ltiIntegration", "administratorPortal", "end2EndExcryption",
            "chatSupport", "translatedCaptions"];
usage_limits = ["maxAssistantsPerMeeting", "maxTimePerMeeting",
                "recordingsCloudStorage"];
plans = ["BASIC", "PRO", "BUSINESS"];
addons = ["hugeMeetings", "translatedCaptions", "phoneDialing"];

plans_prices = [0.0, 15.99, 21.99];
addons_prices = [50.0, 5.0, 100.0];

boolean_usage_limits = [0,0,0];

linked_features = array2d(USAGE_LIMITS, FEATURES, [
    % maxAssistantsPerMeeting
    1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    % maxTimePerMeeting
    1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    % recordingsCloudStorage
```

```
    0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0
]);

plans_features = array2d(PLANS, FEATURES, [
    % BASIC
    1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
    % PRO
    1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0,
    % BUSINESS
    1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0
]);
plans_usage_limits = array2d(PLANS, USAGE_LIMITS, [
    % BASIC
    100, 40, 0,
    % PRO
    100, 1800, 5,
    % BUSINESS
    300, 1800, 5
]);

addons_features = array2d(ADDONS, FEATURES, [
    % hugeMeetings
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    % translatedCaptions
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
    % phoneDialing
    0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0
]);
addons_usage_limits = array2d(ADDONS, USAGE_LIMITS, [
    % hugeMeetings
    1000, 0, 0,
    % translatedCaptions
    0, 0, 0,
    % phoneDialing
    0, 0, 0
]);
addons_usage_limits_extensions = array2d(ADDONS, USAGE_LIMITS, [
    % hugeMeetings
    0, 0, 0,
    % translatedCaptions
    0, 0, 0,
    % phoneDialing
    0, 0, 0
]);
addons_available_for = array2d(ADDONS, PLANS, [
    % hugeMeetings
    1, 1, 1,
    % translatedCaptions
    1, 1, 1,
    % phoneDialing
    0, 1, 1
]);
addons_depends_on = array2d(ADDONS, ADDONS, [
    % hugeMeetings
    0, 0, 0,
    % translatedCaptions
    0, 0, 0,
    % phoneDialing
    0, 0, 0
]);
addons_excludes = array2d(ADDONS, ADDONS, [
    % hugeMeetings
    0, 0, 0,
```

```
    % translatedCaptions
    0, 0, 0,
    % phoneDialing
    0, 0, 0
]);
```

In summary, this hierarchical structure streamlines the AA, since users only have to select one of the "final models" to perform a specific operation. The framework then automatically constructs a complete CSP instance, resolving all dependencies related to the chosen operation and integrating the specific data of the selected iPricing.

4.3 Validation

To validate the proposed framework, we utilized two distinct datasets. The first one originates from the benchmark proposed in [7], which includes a total of 162 pricings that represent the evolution of 30 different real-world SaaS over a period of 5 years (from 2019 to 2024). The second dataset comprises synthetic pricings, which were designed with intentional errors to test the consistency rules enforced by our formalization. Furthermore, all validation experiments were conducted on a MacBook Pro equipped with an 8-core Intel Core i9 processor running at 2.3 GHz and 16 GB of RAM.

A dataset of synthetic, inconsistent pricings

To evaluate the robustness of our consistency validation framework, we constructed a dataset of 20 synthetic pricings designed to simulate real scenarios while deliberately introducing inconsistencies. Each synthetic pricing adheres to the syntax rules of Pricing2Yaml, but exhibits inconsistencies that render them invalid within the iPricings domain, either by preventing the generation of valid subscriptions or by including "dead" pricing elements —e.g. add-ons that cannot be contracted due to restrictions; plans with the same set of features and usage limits, but different prices, etc.

For instance, imagine that Zoom's "translated captions" add-on were marked as available for the BASIC plan, but dependent on the "phone dialing" add-on, which is not offered for such plan. In this case, "translated captions" would also be unavailable for BASIC, despite being incorrectly marked as available. Such logical inconsistencies, while not violating structural rules in Pricing2Yaml, result in pricings that fail to generate the expected configuration space.

This dataset provides a comprehensive testbed for assessing our framework's ability to identify such issues. Detailed descriptions of each synthetic pricing and the specific inconsistencies they are designed to test are included within the laboratory package [8].

Validation of Consistency Rules

For consistency validation, we applied our framework to both datasets, utilizing the *valid-subscription.mzn* model. Our goal was twofold: to ensure that the manually modeled versions of real-world SaaS pricings from [7] adhered to the constraints of the iPricings domain, and to distinguish whether any detected inconsistencies originated from the extracted pricing itself or from errors due to human intervention in the modeling process.

This validation yielded significant findings. In the dataset proposed by the authors of [7], a total of 35 pricings —21.6% of the modeled cases— were found to contain consistency errors. They are detailed as follows:

- 25 linked feature mismatches, where non-zero usage limits were assigned to unavailable features.
- eight incorrect feature definitions, where features were incorrectly represented as numerical values instead of usage limits.

- one temporal error, with a pricing having a future creation date
- one pricing without specific prices (Trustmary 2020), where all plans required contacting sales.

Notably, 34 out of 35 inconsistencies stemmed from errors during the manual transition from web pricings to iPricings, underscoring the risk of human error in this process. To address this, we corrected all 34 inconsistencies originating from human errors, ensuring the dataset aligns with the intended specifications.[3] Additionally, to mitigate similar issues in the future, we have developed a Pricing2Yaml editor within SPHERE, a comprehensive platform for intelligent pricing-driven solutions.[4] The editor provides real-time error detection during the modeling process and offers a live rendering of the pricing as it is constructed/edited, streamlining the development of reliable iPricings.

In contrast, during validation conducted using the synthetic dataset, our solution successfully detected 13 out of the 20 inconsistencies introduced. Notably, all 13 detected cases were directly related to the operations covered by our proposal, whereas the remaining cases were not. Moreover, with a mean time of 147 ms, these results highlight the efficacy of both our approach and the underlying Pricing4TS library, whose Pricing2Yaml parser was able to pre-filter and avoid inconsistencies before execution in MiniZinc, significantly optimizing processing time.

However, the seven undetected inconsistencies involve cases that go beyond the current scope of this work and open opportunities for future research. For instance, our solution does not currently address scenarios where one plan is subsumed by another, i.e. when a plan offers the same features and usage limits as another but at a different price, or when a more expensive plan provides inferior usage limits compared to a cheaper one. Additionally, detecting "dead elements" within pricings —such as add-ons that cannot be contracted due to conflicting dependencies or availability across plans— remains unaddressed. These types of validations require novel operations and constraints that were not within the domain of this study, leaving them as promising directions for further exploration.

Validation of Analytical Operations

For analytical operations, we focused on comparing the results of our framework with manually calculated metrics provided in the benchmark of [7]. Operations such as configuration space computation, identification of the cheapest subscription, and filtering were executed on the benchmark pricings. The outputs of our framework matched the results documented by the authors of [7], with a mean execution time of 327 ms, confirming the correctness of our approach. Additionally, as shown in Tab. 4.2, we present the execution time metrics for each pricing of every SaaS in the dataset of real-world pricings, demonstrating again the efficiency of our proposal.

This validation process has demonstrated how our approach improves the efficiency and reliability of iPricings analysis. The insights derived from iPricings are critical for decision-makers, enabling them to make informed choices based on latent information. Furthermore, a proof-of-concept system implementing our approach is available in SPHERE, showcasing pricing information for all SaaS included in the real-world dataset. For instance, the following link, `http://sphere.score.us.es/pricings/card?name=Buffer`, provides detailed information about Buffer, including the evolution of several key metrics. This demonstrates the practical application and value of our methodology in real-world scenarios.

Replicability and Verifiability For the sake of replicability and verifiability, all the artifacts and datasets generated in this study are available in [8]. This material comprises this technical report, both used datasets, and the necessary scripts and instructions to replicate our experiment.

[3] The corrected real-world pricings dataset is included in the laboratory package [8]
[4] `http://sphere.score.us.es/editor`

SaaS Name	2019	2020	2021	2022	2023	2024
Box	326 ms	306 ms	314 ms	311 ms	308 ms	307 ms
Buffer	296 ms	294 ms	346 ms	327 ms	325 ms	334 ms
Canva	308 ms	322 ms	304 ms	309 ms	295 ms	320 ms
Clickup	305 ms	323 ms	379 ms	374 ms	386 ms	382 ms
Clockify	302 ms	301 ms	297 ms	322 ms	306 ms	329 ms
Crowdcast	-	293 ms	294 ms	298 ms	290 ms	287 ms
Databox	295 ms	290 ms	311 ms	295 ms	722 ms	700 ms
Deskera	-	-	295 ms	286 ms	293 ms	322 ms
Dropbox	-	-	305 ms	315 ms	308 ms	313 ms
Evernote	288 ms	305 ms	300 ms	299 ms	294 ms	293 ms
Figma	290 ms	287 ms	299 ms	294 ms	302 ms	316 ms
Github	308 ms	317 ms	441 ms	453 ms	1233 ms	1337 ms
Hypercontext	-	-	301 ms	294 ms	293 ms	302 ms
Jira	286 ms	305 ms	295 ms	298 ms	304 ms	305 ms
Mailchimp	307 ms	296 ms	328 ms	334 ms	327 ms	331 ms
Microsoft	306 ms	318 ms	323 ms	323 ms	290 ms	307 ms
Notion	-	-	290 ms	294 ms	302 ms	313 ms
Openphone	-	294 ms	310 ms	319 ms	432 ms	443 ms
Overleaf	297 ms	290 ms	290 ms	286 ms	288 ms	289 ms
Planable	299 ms	292 ms	293 ms	301 ms	307 ms	318 ms
Postman	-	345 ms	345 ms	374 ms	941 ms	818 ms
Pumble	-	-	291 ms	290 ms	293 ms	295 ms
Quip	292 ms	302 ms	292 ms	294 ms	295 ms	289 ms
Salesforce	327 ms	322 ms	348 ms	833 ms	574 ms	8311 ms
Slack	291 ms	311 ms	-	-	302 ms	332 ms
Tableau	285 ms	298 ms	308 ms	305 ms	310 ms	321 ms
Trustmary	-	-	293 ms	290 ms	294 ms	297 ms
Userguiding	-	301 ms	294 ms	296 ms	298 ms	315 ms
Wrike	410 ms	406 ms	335 ms	336 ms	372 ms	372 ms
Zapier	292 ms	305 ms	-	293 ms	301 ms	324 ms

Table 4.2: Evaluation of SaaS response times (in milliseconds) for the years 2019 to 2024. Rows with ”-” indicate missing data for specific years.

Chapter 5

Conclusions and Future Work

In this work, we have formalized iPricings as Constraint Satisfaction Problems (CSPs), introducing a generic restriction, π, that encapsulates all the constraints inherent to pricings. This formalization allows for the analysis of multiple pricings without redefining the CSP for each case, as π can be paired with a data file specifying the details of a particular pricing. This approach enables the extraction of specific insights from any given pricing while maintaining the same underlying CSP structure. Furthermore, we have proposed a set of analysis operations based on this formalization, like the cardinality of the configuration space or the optimization of subscriptions according to cost or feature requirements.

As part of our validation efforts, we i) identified and corrected inconsistencies in a real-world SaaS pricing dataset [7], enhancing its reliability for future studies[1]; ii) developed a synthetic dataset specifically designed to benchmark the detection of pricing inconsistencies, offering a robust foundation for evaluating and improving future contributions in this domain; iii) introduced a Pricing2Yaml editor within SPHERE,[2] designed to improve the transition from web pricings to iPricings by reducing human error through real-time error detection and live rendering of pricings during their construction; and iv) also integrated within this platform a solution based on our approach: the pricing cards[3].

Looking ahead, this work opens promising avenues for extending the automated analysis of iPricings, particularly in the context of SaaS pricings. Future efforts could prioritize enhancing the explainability of the proposed operations, offering actionable insights to guide developers and pricing strategists towards better-informed decisions. Furthermore, building on our validation findings that highlight the significant risk of human error during the manual transformation from web pricings to iPricings, a key direction for future work lies in developing automated solutions to streamline this transition. Initial steps in this direction have already been explored in [3], where approaches leveraging large language models (LLMs) have been proposed to support this transformation process, providing a foundation for further refinement and expansion.

By embedding these techniques into practical tools and addressing these future avenues, we aim to establish iPricings as a cornerstone for pricing-driven variability management, meeting the evolving demands of modern SaaS systems.

[1] The new version of the industry SaaS pricings dataset is available on our labpack [8]
[2] http://sphere.score.us.es/editor
[3] Sample pricing card: http://sphere.score.us.es/pricings/card?name=Buffer

Bibliography

[1] Benavides, D., Trinidad, P., Ruiz-Cortés, A.: Automated reasoning on feature models. In: International Conference on Advanced Information Systems Engineering. pp. 491–503. Springer (2005)

[2] Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated Analysis of Feature Models 20 Years Later: A Literature Review. Information Systems **35**(6), 615 – 636 (2010)

[3] Cavero, F.J., Alonso, J.C., Ruiz-Cortés, A.: From Static to Intelligent: Evolving SaaS Pricing with LLMs. In: Proceeding of the International Conference on Service Oriented Computing, (ICSOC). 1st International Workshop on Service-Oriented Computing for Artificial Intelligence, (SOC4AI), Springer LNCS (2024)

[4] García-Fernández, A., Parejo, J.A., Ruiz-Cortés, A.: Pricing-driven Development and Operation of SaaS: Challenges and Opportunities. In: Actas de las XIX Jornadas de Ciencia e Ingeniería de Servicios (JCIS). SISTEDES (2024)

[5] García-Fernández, A., Parejo, J.A., Ruiz-Cortés, A.: Pricing4SaaS: Towards a pricing model to drive the operation of SaaS. In: Intelligent Information Systems, - CAiSE Forum, Proceedings. Lecture Notes in Business Information Processing, vol. 520, pp. 47–54. Springer (2024)

[6] García-Fernández, A., Parejo, J.A., Trinidad, P., Ruiz-Cortés, A.: Towards Pricing4SaaS: A Framework for Pricing-Driven Feature Toggling in SaaS. In: Web Engineering. 24th International Conference, ICWE. pp. 389–392. Springer (2024)

[7] Garcia-Fernández, A., Parejo, J.A., Cavero, F.J., Ruiz-Cortés, A.: Racing the market: An industry support analysis for pricing-driven devops in SaaS. In: Proceeding of the International Conference on Service Oriented Computing, (ICSOC). (In Press) (2024). https://doi.org/10.48550/arXiv.2409.15150

[8] García-Fernández, A., Parejo, J.A., Trinidad, P., Ruiz-Cortés, A.: CAISE'25 Laboratory Package (2024), https://github.com/isa-group/SaaS-analysis/tree/CAISE'25-Research-Track, este es el labpack temporal. Cuando cerremos la versión definitiva, publicaré en zenodo y cambio la cita

[9] Jiang, Z., Sun, W., Tang, K., Snowdon, J., Zhang, X.: A pattern-based design approach for subscription management of software as a service. 2009 Congress on Services - I pp. 678–685 (2009)

[10] Müller, C., Gutierrez Fernandez, A.M., Fernandez, P., Martin-Diaz, O., Resinas, M., Ruiz-Cortes, A.: Automated Validation of Compensable SLAs. IEEE Transactions on Services Computing (2018)

[11] Müller, C., Resinas, M., Ruiz-Cortés, A.: Automated Analysis of Conflicts in WS–Agreement. IEEE Transactions on Services Computing **7**(4), 530–544 (2014)

[12] Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard cp modelling language. In: International Conference on Principles and Practice of Constraint Programming. pp. 529–543. Springer (2007)

[13] Ter Hofstede, A.H.M., Proper, H.A.: How to Formalize It? Formalization Principles for Information System Development Methods. Information and Software Technology **40**, 519–540 (1998)

[14] Trinidad, P., Ruiz-Cortés, A., Benavides, D.: Automated Analysis of Stateful Feature Models, pp. 375–380. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)