From 4GL Spreadsheet Computations to Constraint Model Definitions - A Development Process

Boi Schaefer[1,†], Lothar Hotz[1,*,†] and Kirsten David[2]

[1]Hamburger Informatik Technologie-Center e.V., Vogt-Kölln-Straße 30, 22527 Hamburg
[2]Universität Bielefeld, Universitätsstraße 25, 33615 Bielefeld

Abstract

In this paper, we present an approach for mapping variables and equations given in a tabular application of a 4GL spreadsheet (fourth generation programming language) to a constraint model. We start with a spreadsheet given for a specific application in the area of modernization of buildings. The spreadsheet computes for landlords and tenants the increase of the rent after a modernization is done. These computations shall be part of a platform that enables computation and negotiation of building modernization endeavor. This approach is particularly relevant for configuration systems where domain experts typically express configuration knowledge through spreadsheets, and constraint-based configuration platforms require declarative constraint models. Our development process bridges this gap by providing a systematic methodology to transform 4GL spreadsheet computations into maintainable constraint models, enabling better integration of domain expertise into configuration platforms.

Keywords

4GL Tables, PyChoco, Constraint Model, OR-Tools, SCREAMER

1. Introduction

Constraint models enable computations by declaratively specifying variables with their domains and constraints between them, which are then processed by a constraint solver tool [1]. An application problem is defined through variables with appropriate domains and constraints between the variables as a Constraint Satisfaction Problem (CSP) [2]. One way to provide the necessary computations of an application task to developers of a constraint model is to give a formal representation of variables and equations in a spreadsheet from a fourth-generation language (4GL) [3][1] such as Microsoft EXCEL. This challenge is particularly prevalent in configuration systems, where domain experts often express configuration knowledge through familiar spreadsheet interfaces, while the underlying platform requires formal constraint models to enable automated reasoning, optimization, and validation. Thus, in a platform, the spreadsheet interface is replaced by a user interface and the computations are done with a constraint system in the backend.

While several approaches exist for converting spreadsheets to other formats, existing research has predominantly focused on data migration and format transformation rather than constraint model generation. For example, Harris and Gulwani [4] develop methods for automatically restructuring spreadsheet tables, while Shigarov et al. [5] present rule-based approaches for converting arbitrary spreadsheet tables into relational database formats. These approaches demonstrate sophisticated data transformation capabilities but primarily address format conversion and data migration scenarios, preserving data content while losing the underlying computational logic embedded in spreadsheet formulas. Direct code generation techniques can maintain computational aspects but sacrifice the

ConfWS'25: 27th International Workshop on Configuration, Oct 25–26, 2025, Bologna, Italy
*Corresponding author.
†These authors contributed equally.
boi.schaefer@hitec-hamburg.de (B. Schaefer); lothar.hotz@hitec-hamburg.de (L. Hotz); kirsten.david@uni-bielefeld.de (K. David)
https://hitec-hamburg.de (B. Schaefer); https://kogs-www.informatik.uni-hamburg.de/~hotz/ (L. Hotz); https://www.uni-bielefeld.de/fakultaeten/rechtswissenschaft/forschung/forschungsstellen/fir/intelmod/ (K. David)
0000-0001-7370-7726 (L. Hotz)

[1]https://en.wikipedia.org/wiki/Fourth-generation_programming_language

declarative nature essential for constraint-based reasoning. Rule-based systems, while capable of capturing logical structures, struggle with complex mathematical relationships and multidirectional computation capabilities provided by constraint-based configuration systems.

Converting spreadsheet computations to constraint models presents unique challenges that extend beyond traditional data transformation approaches. Main questions arise such as: What are the variables, domains, and constraints? How are variables grouped? What are input variables? Which are computed? How are the computations being represented, with integer or mixed-integer programming? A further aspect is the dynamic generation of the CSP, depending on the input values of the user which might occur in the spreadsheet. Additionally, constraint solvers are typically back-end libraries without user interfaces, which requires integration with front-end components for user interaction.

In this paper, we present our approach of a development process (Section 3) that leads from a spreadsheet to a constraint model. First, we describe in Section 2 our application in the area of building modernization. This application is used throughout the paper for demonstrating our approach, especially, because the application comes with a spreadsheet developed by a domain expert. This task is also relevant for configuration tasks as the spreadsheet computation may also be used for representing configuration knowledge, as well as the dynamic aspect of the CPS generation relates to configuration. Then, we elaborate a pre study with a programming language and constraint solver that enable fast developments. This step has answered the question, if in principle the spreadsheet can be mapped to a constraint solver (see Section 4). In the next step, we select an appropriate constraint solver (see Section 5). The modeling and implementation was done with the selected OR-Tools solver and we analyzed the use of integer modeling vs. mixed-integer programming (see Section 6). Section 7 presents the overall architecture and interfaces, as Section 8 discusses the approache and Section 9 concludes.

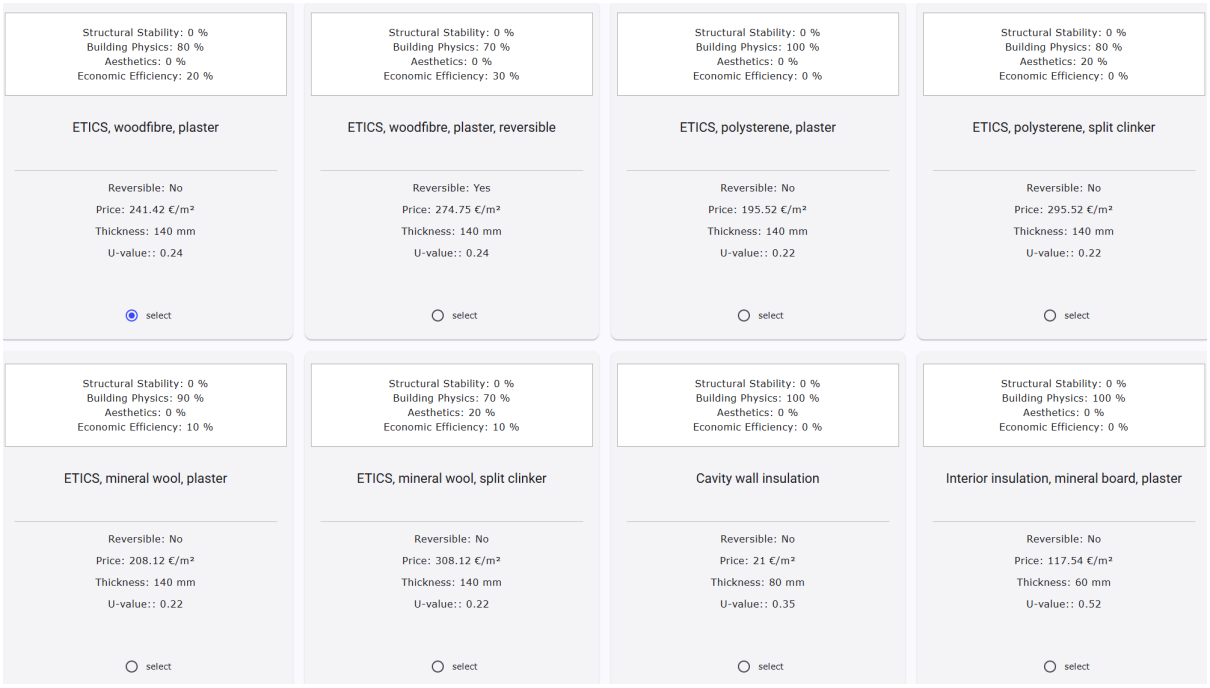


Figure 1: Catalog of construction measures for modernization with percentage indications.

2. Application Description

This research is part of the Intelligent Modernization Platform (IntelMOD) project, which aims to offer a tool to tenants and landlords to initiate modernization negotiations based on eight building functions: Stability (Standsicherheit), Moisture Protection (Feuchteschutz), Thermal Insulation (Wärmeschutz), Sound Insulation (Schallschutz), Fire Protection (Brandschutz), Daylight Access (Tageslicht), Aesthetics

(Ästetik), and Ecological and Economical Efficiency (Wirtschaftlichkeit) which the so called *Functional Cost Splitting (FK)* provides [6]. The FK is a recommended course of action for determining fair and traceable rent adjustments following energy-efficient modernization. It can be used in practice by landlords and tenants as a transparent basis for communication and calculation for object- and even measure-specific rent increases for modernizations to avoid or resolve disputes while simultaneously optimizing the degree of sustainability of the relevant measures. With the framework of FK, the essential step towards organizing the necessary information, data, and regulations of modernization knowledge has already been taken.

2.1. Functional Cost Splitting

The Functional Cost Splitting of the IntelMOD platform determines the rent increase relevant and non-rent increase relevant cost components of modernization measures according to German rental law in three successive steps.

2.1.1. Step I: Project Data and Weighting of Improvements Planned with the Measure

The first step captures project data and defines a percentage weighting of the eight basic functions of a building component: Stability, Moisture Protection, Thermal Insulation, Sound Insulation, Fire Safety, Daylight Access, Aesthetics, and Ecological and Economical Efficiency. Initially, basic building data including total floor area of the building, the size of the affected apartment and total costs of the modernization measure are entered. The system distinguishes between three states: the ACTUAL state before modernization, the planned MOD state of the modernization measure, and the resulting NEW state after implementation.

Function Level Structure

ACTUAL State - Function Level 1 (Input): At the first level, four main functions are weighted: Stability, Building Physics, Aesthetics, Ecological and Economical Efficiency

ACTUAL State - Function Level 2 (automatically calculated): Computations for building physics weighting which is automatically divided equally among the five sub-functions: Moisture Protection, Thermal Insulation, Sound Insulation, Fire Safety, and Daylight Access.

MOD State - (Input): For modernization planning, all eight individual functions are directly weighted according to specific modernization objectives.

NEW State - (automatically calculated): The target state is calculated as a weighted average between ACTUAL and MOD state for all eight individual functions and represents the intended function distribution after modernization.

2.1.2. Step II: Building Data and Assessment of Improvements Actually Achieved with the Measures

In the second step, for each of the eight individual functions, it is entered whether an actual improvement is achieved through the planned modernization measure. This assessment is made through binary Yes/No decisions for each function as further input variables.

2.1.3. Step III: Building Data and Assessment of the Fulfillment Grade (Under-fulfillment) of Functions of the Existing Building Component for Estimation of Required Maintenance Costs

The third step captures the current condition of the existing building component through detailed assessment. For each of the eight individual functions, the fulfillment grade of the ACTUAL state is

determined by entering existing damages and their extent and effects on the respective function as next input variables. The completion grade assessment is performed by entering percentages of the damage condition in relation to the affected area and functional impairment. The system automatically calculates the maintenance-relevant cost component that is attributable to the repair of existing defects. This calculation is only done for the selected damage assessment, that is, the fields in the spreadsheet are only computed if other fields are selected by the user (*dynamic field activation*), hence, dynamic constraint creation (see below).

2.1.4. Final Cost Calculation

The system automatically calculates all relevant cost indicators based on the inputs from the three steps. From the determined rent increase relevant cost component, the annual and monthly cost components are calculated for both the entire building and per square meter and for the affected apartment. The automatic calculations include proportional distribution by floor area and consider the function weightings and fulfillment grades determined in the previous steps. The system ensures that rent increase relevant components and non-rent increase relevant components are correctly separated.

2.1.5. Representation of the FK

The FK was developed in a 4GL spreadsheet with the tool EXCEL by Kirsten David [6]. Users can use these spreadsheet for computing the modernization cost as depicted above. There by, they provide the input variables. Through 4GL equations such as `=IF('Bewertung Verbesserung NEU'!L26='Bewertung Verbesserung NEU'!J24; 'Definition Funktions-Soll NEU'!N41; "keine Verbesserung")` or `=(R25+R33)/2` computations are executed by the spreadsheet. The MOD state is depicted with weighted values (s.a). Of course those might be difficult to provide, hence, a catalog of construction measures for modernization is given to the user where each construction measure maps to specific weights for the functions (see Figure 1). In total, the calculations of the FK are specified as requirements using EXCEL spreadsheet tables. The EXCEL spreadsheet contains not only prescribed equations but also configuration-dependent formulas. Different building configurations require different calculation paths. For instance, buildings with certain damage patterns activate additional worksheets with specific formulas, while buildings without those damages skip these calculations entirely. This conditional logic creates a dynamic calculation structure that varies based on the input configuration.

3. Conceptual Approach

For an application in a backend of an Internet platform the spreadsheet cannot be used and has to be converted into a program. We did not consider a procedural program in a high-level programming language as implementation target, because we do not want to loose flexibility in the computations as well as ensure easy maintenance. Instead, we choose constraint programming as an implementation approach that ensures a declarative description of the problem through variables with appropriate domains, constraints, and a hidden algorithm for solving the problem implemented in a solver. Thus, this approach separates the knowledge of the calculation from the actual computation itself. Further advantages when using domains, i.e., intervals, ranges can be taken into account for input variables (e.g., weight ranges) which are computed by the solver (leading to, e.g., ranges of rent increase). Also the multidirectional feature of constraint solvers is enabled by this approach and will be used in the future for reverse computing from potential rent increase to needed modernisation construction measures.

The steps of our approach are: pre study, tool selection, CSP implementation, architecture, and user interface. They are presented in the following sections.

```
(defstruct (IST-funktionen-gewichtung-bestehendes-bauteil-fkt-ebene-1
            (:conc-name fgbb-IST-fkt-1-)
            (:print-function print-fgbb-IST-fkt-1))
  standsicherheit
  bauphysik
  aesthetik
  wirtschaftlichkeit
  summe)

(defun create-a-bestehendes-bauteil-IST-fkt-ebene-1 ()
  (let ((ss (an-integer-betweenv 0 *upperbound* "IST-fkt-1-ss"))
        (bp (an-integer-betweenv 0 *upperbound* "IST-fkt-1-pb"))
        (ae (an-integer-betweenv 0 *upperbound* "IST-fkt-1-ae"))
        (wk (an-integer-betweenv 0 *upperbound* "IST-fkt-1-wk"))
        (su (an-integer-betweenv 0 *upperbound* "IST-fkt-1-su")))
    (assert! (=v su (+v ss bp ae wk)))
    (assert! (=v su *upperbound*))
    (make-IST-funktionen-gewichtung-bestehendes-bauteil-fkt-ebene-1
     :standsicherheit ss
     :bauphysik bp
     :aesthetik ae
     :wirtschaftlichkeit wk
     :summe su)))
```

Figure 2: SCREAMER/COMMON LISP implementation for the IST/ACTUAL functions of Level 1. A `defstruct` groups variables of one row describing the actual values. A function (`create-a-bestehendes-bauteil-IST-fkt-ebene-1`) defines constraint variables and constraints of that row.

4. Pre Study

We first developed a prototypical implementation of the Functional Cost Splitting (FK) in COMMON LISP [7] using the constraint system SCREAMER [8]. Creating a machine-readable structure, the FK had to be converted into a digital and machine-readable format. For this purpose, the existing process was analyzed and translated into standardized constraints, which enable formal computation. A prototype for the FK automation was implemented based on constraints in SCREAMER, successfully realizing significant portions of the FK.

In scientific processing and optimization, spreadsheet data can often be represented using constraints. This method allows for an explicit modeling of relationships between spreadsheet expressions and variables. Variables in this context refer to parameters or functions dictated by the problem structure (e.g., weights for Stability, Thermal Insulation). Constraints, on the other hand, represent the conditions that these variables must satisfy (e.g., the sum of all values must be exactly 100%).

The grouping of variables depends on the characteristics of the problem to ensure an overview, efficient modeling, traceability, and maintainability. The use of large language models (LLMs) for creating a constraint model from the spreadsheet was not considered sustainable due to deficiencies in maintainability and scalability of the resulting code, which contained, e.g., number-based filed names such as $R23$.

To address these issues, the SCREAMER COMMON LISP implementation was chosen. This provides a powerful platform for realizing constraint models (see Figure 2 for an example for the implementation of ACTUAL functions of Level 1). A test-driven approach was employed to ensure the correctness of the implementation – particularly in cases involving mutual computations between variables.

The developed model focuses on the application of the FK and includes:

Variables: Functional parameters such as weights for Stability, Thermal Insulation, etc., i.e., the fields in the 4GL spreadsheet.

Domains: Value ranges of these variables (e.g., 0% to 100%).

Constraints: Specifications and technical requirements (e.g., the sum of all values must be exactly 100%).

The advantages of constraint modeling lie in its efficiency for solving complex problems, its adaptability, as well as the comprehensibility and verifiability of the results.

As result of the pre study, we developed a semantically meaningful grouping of variables which directly corresponds to rows in the spreadsheet. Furthermore, the identification of the input variables, constraints, and output variables was done. Through a test-driven approach, we could show the computability of the spreadsheet through a constraint solver.

5. Tool selection

We did not use the COMMON LISP as a basis for the platform implementation, because of less available programming skills in this sector. Instead, we used a more known programming language, i.e., PYTHON.

We compared the CSP solvers PyChoco [9][2], Pyomo[3], and OR tools[4]. These three solvers were selected for comparison to evaluate different approaches in constraint and optimization programming. PyChoco represented the category of specialized constraint solvers with established Java foundation and PYTHON interface. Pyomo stood for flexible mathematical optimization frameworks that support various solver backends. OR-Tools represented integrated optimization platforms that combine multiple solver types (CSP, MIP, LP) in a unified environment. Through this deliberate selection, we could systematically evaluate three different philosophies according to the defined criteria and identify the optimal solution for our FK system.

Comparison criteria are:

1. Age and Latest Version: Evaluating the maturity and current updates of each solver is essential to ensure reliability and sustainability.
2. Programming Language Support: The availability of interfaces in desired programming languages (e.g., PYTHON) was a critical factor for ease of integration.
3. Embedded vs. External Solver Connection: Whether the solver can be seamlessly integrated into existing workflows or requires external setup.
4. Versatility and Range of Functions: Assess the ability to handle various problem types, such as CSP, linear programming (LP), mixed-integer programming (MIP), and routing problems.
5. Documentation and Community Support: The availability of comprehensive documentation and active developer communities for troubleshooting and updates.
6. Literature and Resources: Access to academic papers, tutorials, and case studies that support learning and implementation.

OR tools emerged as the preferred choice cause of several advantages:

1. Regular Updates and Active Maintenance: The tool receives consistent updates, ensuring it remains current and reliable.
2. Multi-Language Support: Particularly strong support for PYTHON, which aligns with modern development trends and ease of use.
3. Powerful Integrated Solvers: OR tools offer robust solvers directly embedded within the platform, streamlining the problem-solving process.

[2] https://github.com/choco_solver/pychoco
[3] https://pyomo.readthedocs.io/en/latest/
[4] https://github.com/google/or-tools

4. Versatility in Problem Types: It supports a wide range of optimization problems, including CSP, LP, MIP, and routing challenges.
5. Extensive Documentation: Comprehensive documentation is available to guide users through implementation and troubleshooting.
6. Large Developer Community: A vibrant developer community provides active support and shares knowledge, enhancing the tool's ecosystem.
7. Special Recognition: OR tools have received significant recognition in the field, including multiple wins in the MiniZinc Challenges, a prestigious competition for CSP solvers.

In conclusion, after evaluating these criteria, OR tools were selected due to their comprehensive functionality, strong PYTHON support, active development, and proven track record in solving complex optimization problems.

```python
    class ISTFunktionenGewichtungBestehendesBauteilFktEbene1:
        """
        Class representing the existing functions weighting for an existing
            building component at function level 1.
        """

        def __init__(
        self,
        standsicherheit,
        bauphysik,
        aesthetik,
        wirtschaftlichkeit,
        summe,
        ):
        # Initialize the attributes with the provided variables
        self.standsicherheit = standsicherheit
        self.bauphysik = bauphysik
        self.aesthetik = aesthetik
        self.wirtschaftlichkeit = wirtschaftlichkeit
        self.summe = summe
```

Figure 3: Part 1/2: Class definition for grouping variables. See Figure 2 for the COMMON LISP implementation.

6. OR-Tools Implementation

6.1. Technical Implementation as Constraint Model

We implemented the FK as a CSP with OR Tools. The constraint model comprises 23 fixed input fields, hence, variables, for Step I and II, complemented by a dynamic number of input fields ranging from 0 to 330 variables depending on the extent of damage assessment selected in Step III. The constraint structure follows a similar pattern with 130 fixed constraints and an additional 0 to 288 dynamic constraints that scale with the complexity of the damage evaluation process.

The primary constraints ensure mathematical consistency by requiring that function weightings in each state sum to exactly 100% and that fulfillment grades remain within the valid range of 0% to 100%. Other constraints are related to mathematical computations for the rent increase. The model generates 8 primary output variables as illustrated in the corresponding Figure 6, representing the final cost allocation and rent increase calculations derived from the functional cost splitting methodology.

```python
def create_a_bestehendes_bauteil_IST_fkt_ebene_1(model):
        """
        Creates an existing building component at function level 1 (IST FKT 1).

        Parameters:
        - model: The CP-SAT model where variables and constraints are added

        Returns:
        - An instance of ISTFunktionenGewichtungBestehendesBauteilFktEbene1
            containing the variables for IST FKT 1
        """
        # Define integer variables for each function, ranging from 0 to 100
        ss = model.NewIntVar(0, 100, "IST_fe1_ss")
        bp = model.NewIntVar(0, 100, "IST_fe1_bp")
        ae = model.NewIntVar(0, 100, "IST_fe1_ae")
        wk = model.NewIntVar(0, 100, "IST_fe1_wk")
        su = model.NewIntVar(0, 100, "IST_fe1_su")

        # Add constraints to ensure the sum of all function values equals the
            total sum variable
        model.Add(ss + bp + ae + wk == su)
        # The total sum should be exactly 100 (percent)
        model.Add(su == 100)

        # Return an instance of the IST FKT 1 class with the defined variables
        return ISTFunktionenGewichtungBestehendesBauteilFktEbene1(ss, bp, ae, wk,
             su)
```

Figure 4: Part 2/2: Modeling variables (lines 12 to 16) and two constraints (lines 19 and 21) for the IST/ACTUAL functions of Level 1. See Figure 2 for the COMMON LISP implementation.

The constraint variables are grouped into classes, which typically correspond to rows in the FK spreadsheets. Some classes for Step I are `Building`, `ISTFunktionenGewichtungBestehendesBauteilFktEbene1`, `ISTFunktionenGewichtungBestehendesBauteilFktEbene2`, and `MODFunktionenGewichtungBestehendesBauteilFktEbene1`. Figures 3 and 4 provide the model for class `ISTFunktionenGewichtungBestehendesBauteilFktEbene1`, which combines all variables, their domains, and two constraints for the four weighted functions of the ACTUAL State Function Level 1 which is equivalent to the implementation Figure 2 shows.

6.2. Number Representation

The requirements have demonstrated that the domains of the constraint variables exhibit diverse data types. Percentages associated with FK functions are represented using integer values, as decimal places would not make sense here since the user is intended to estimate these percentages subjectively. Conversely, real estate calculations, including rent increases, necessitate real numbers.

In selecting a constraint tool, one has the option between utilizing integer or mixed-integer programming for processing integers and/or real values. Initially, we employ the integer algorithm, which required a systematic scaling procedure for decimal values. Monetary values (such as modernization costs) were multiplied by a scaling factor of 10,000 to ensure sufficient precision in the cent range. Percentage values (such as under-fulfillment) were scaled by a factor of 100 to obtain one decimal place (e.g. 42.7% becomes 4270 in the solver).

Before constraint solving, all necessary input values are scaled accordingly. Calculations proceed

within integer domain, with results subsequently scaled back for output purposes. This procedure ensures both the required numerical precision and compatibility with integer constraint solvers.

Precise determination of calculations for the first two table sheets was achieved after incorporating rounding in the EXCEL calculation for specific test cases.

While integer programming convinces through deterministic results and optimized solver performance, it requires complex scaling procedures for decimal values. Mixed Integer Programming (MIP) offers advantages through direct processing of both integer and continuous variables, thereby avoiding scaling artifacts and simplifying the modeling approach. For future extensions of the FK system, particularly when integrating optimization objectives and interval inputs, MIP could represent a more efficient alternative.

Figure 6 presents an example of the final result of the increase in rent when modernizing.

6.3. Dynamic CSP Generation

As discussed in Section 2, some fields of the spreadsheet are dynamically activated or computed when other fields are filled out. Hence, the constraint satisfaction problem is not the same for each building. Depending on the damage assessment, more or less variables and constraints are present. For example, if moisture damage is identified, additional variables for moisture protection calculations and their corresponding constraints are included in the CSP model. Hence, depending on the input values given by the user, the variables and constraints are selected by a generation module, and the constraint model is created and solved.

6.4. Optimization Aspects

Beyond constraint satisfaction, our approach naturally supports optimization objectives that extend the capabilities of the original spreadsheet implementation. The FK calculations can be enhanced to optimize for various criteria, including cost minimization where the system identifies the minimum cost configuration that achieves required function improvements, and benefit maximization where function improvements are maximized within given budget constraints. The constraint framework also accommodates multi-objective optimization scenarios that balance cost considerations, function improvements, and sustainability metrics simultaneously. Furthermore, Pareto optimization capabilities enable the identification of trade-off solutions between conflicting objectives, providing decision makers with a comprehensive view of available alternatives.

The constraint model structure facilitates these optimization extensions through its clear separation of variables, constraints, and objective functions. This architectural separation enables straightforward integration of optimization goals without requiring fundamental changes to the underlying constraint model. The declarative nature of the constraint representation allows for dynamic objective function specification, where different optimization criteria can be applied to the same underlying model based on user preferences or specific modernization scenarios. These optimization capabilities represent a significant advancement over the original spreadsheet approach, which is limited to single-point calculations and cannot explore solution spaces or identify optimal configurations.

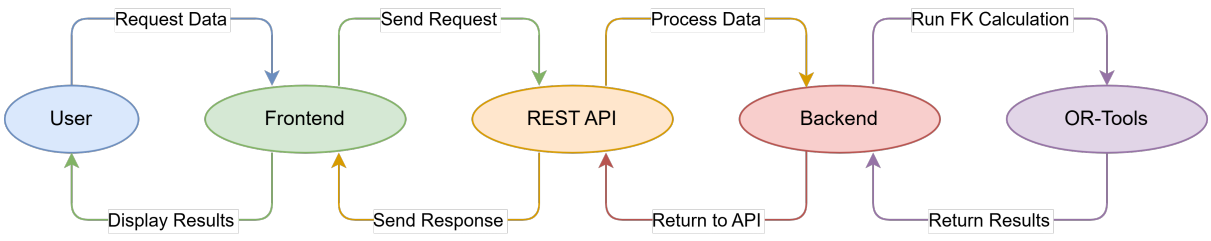


Figure 5: Architecture, including the constraint solver in the backend.

7. Architecture and User Interface

System components divide into the main components:

Front-end: User interface for entering and displaying results. Sends REST requests to the back-end.

REST API: Interface between the front and back end. Defines endpoints for data transfer.

Back-end with OR tools: Processes requests and solves CSP models. Implements functional cost splitting.

The interface data is modeled in the JSON format where key value pairs directly map to variables of the CSP.

The following interactions between the components occur (see Figure 5):

- The User sends request data ("Request Data") via the web client of the user to the Frontend.
- The Front-end processes the request and sends it ("Send Request") to the REST API.
- The REST API receives the request and forwards the data ("Process Data") to the Back-end for processing.
- The Back-end initiates the Functional Cost Splitting (FK) calculation ("Run FK Calculation") and calls OR-Tools for this purpose.
- OR-Tools performs the constraint programming calculations and returns the results ("Return Results") back to the Back-end.
- The Back-end processes these results and forwards them ("Return to API") to the REST API.
- The REST API sends the response ("Send Response") to the Front-end.
- The Front-end prepares the data and displays the results ("Display Results") to the User.

An ontology (not depicted, see [10]) specifies the necessary classes to model the knowledge database, covering legal, construction-related, and FK-related aspects. The FK calculations were provided to the front-end via an API and corresponding interfaces were defined.

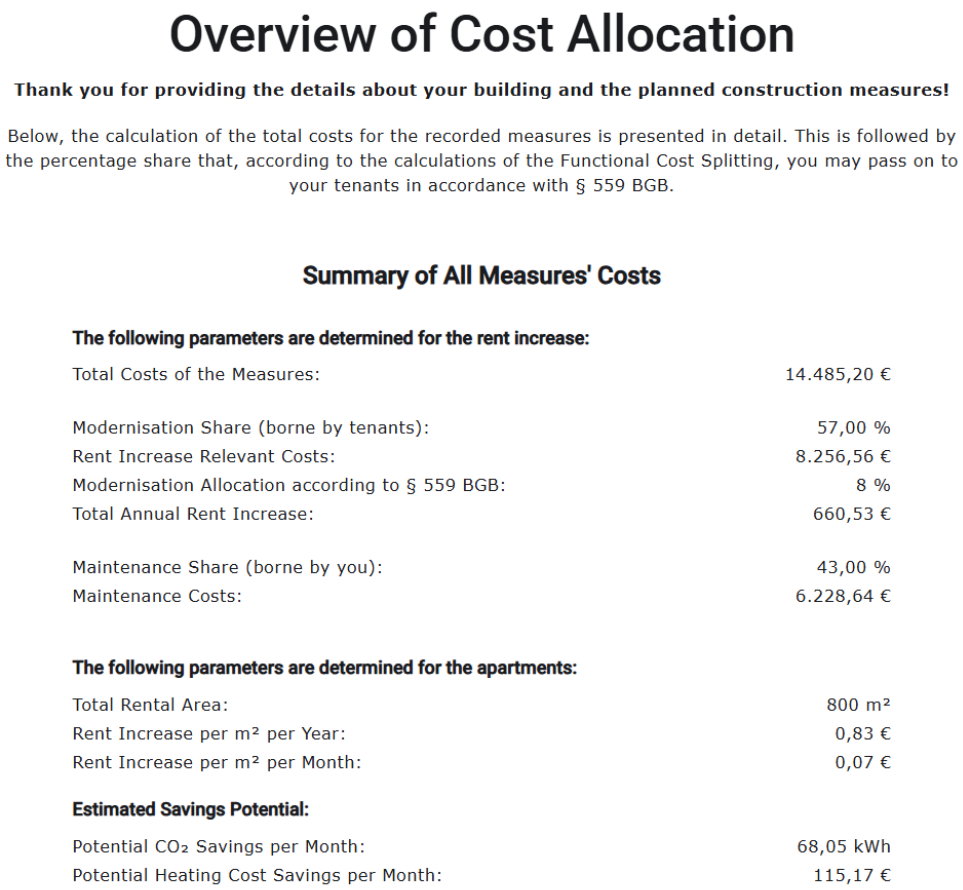
A prototypical user interface enables the entry of building-specific data, the cost of modernization, as well as the input of the weightings for the functions (see Figure 1). The prototype is available on the Internet[5].

8. Advantages and Limitations

Our constraint-based approach offers several significant advantages over alternative methods for converting spreadsheet computations to executable models. The systematic mapping process preserves the semantic structure of the original spreadsheet, maintaining domain expert knowledge intact throughout the transformation. Unlike procedural implementations that hardcode computational logic, constraint models provide a declarative nature that separates problem description from solution algorithms, enabling greater flexibility and maintainability. The declarative approach naturally supports multi-directional computation capabilities, allowing for reverse calculations such as determining required modernization measures from desired rent increase level. Additionally, the constraint framework seamlessly integrates optimization objectives, enabling extensions for cost minimization, benefit maximization, and multi-objective optimization scenarios. The dynamic CSP generation capability effectively handles conditional logic and varying problem complexity based on input configurations, adapting the constraint model size and complexity to the specific building assessment requirements.

However, the approach also presents several limitations that must be acknowledged. The development process requires substantial manual analysis and modeling effort, making it labor-intensive compared to an automated conversion tools, e.g., with LLMs. Effective variable grouping and constraint identification

[5] https://mieter.intelmod.hitec-hamburg.org/



Overview of Cost Allocation

Thank you for providing the details about your building and the planned construction measures!

Below, the calculation of the total costs for the recorded measures is presented in detail. This is followed by the percentage share that, according to the calculations of the Functional Cost Splitting, you may pass on to your tenants in accordance with § 559 BGB.

Summary of All Measures' Costs

The following parameters are determined for the rent increase:

Total Costs of the Measures:	14.485,20 €
Modernisation Share (borne by tenants):	57,00 %
Rent Increase Relevant Costs:	8.256,56 €
Modernisation Allocation according to § 559 BGB:	8 %
Total Annual Rent Increase:	660,53 €
Maintenance Share (borne by you):	43,00 %
Maintenance Costs:	6.228,64 €

The following parameters are determined for the apartments:

Total Rental Area:	800 m²
Rent Increase per m² per Year:	0,83 €
Rent Increase per m² per Month:	0,07 €

Estimated Savings Potential:

Potential CO_2 Savings per Month:	68,05 kWh
Potential Heating Cost Savings per Month:	115,17 €

Figure 6: Result of the computations providing the increase per m^2 per month and further information in 8 variables (the two "total" variables are input variables and the last two are not computed by the CPS).

demand deep understanding of both the application domain and constraint programming principles, limiting the approach's accessibility to domain experts without technical programming knowledge. The resulting system's performance and capabilities are fundamentally constrained by the chosen constraint solver's limitations and computational efficiency.

When compared to alternative approaches, our method provides better maintainability and flexibility than direct code generation but requires more initial development investment. Compared to retaining the original spreadsheet format, our approach enables integration into larger software systems and supports advanced optimization capabilities, though it sacrifices the immediate usability that domain experts experience with familiar spreadsheet interfaces.

9. Conclusion

The paper presents a development process for a constraint model starting from a 4GL spreadsheet. We use this process to implement computations for a building modernization platform with constraints. Main subtasks are the identification of the variables and their domains as well as the constraints. Furthermore, grouping of variables and constraints in rows of the spreadsheet facilitates a transparent implementation. The computations are based on integer and mixed-integer programming. A dynamic aspect of the spreadsheet was mapped onto dynamic constraint model generation, depending on the user's input. Finally, we could completely map the spreadsheet to the constraint model and solve

modernization related rental increases. Further work will encompass the usage of ranges for the input variables.

Acknowledgments

This research was funded by the Federal Ministry for Economic Affairs and Climate Action (Bundesministerium für Wirtschaft und Klimaschutz), Germany, within the "Joint project: EnOB: IntelMOD - Intelligent modernization platform based on functional cost splitting; sub-project: Infrastructure of the modernization platform" ("Verbundvorhaben: EnOB: IntelMOD - Intelligente Modernisierungsplattform auf Basis des Funktionalen Kostensplittings; Teilprojekt: Infrastruktur der Modernisierungsplattform."), FKZ 03EN1094D.

Declaration on Generative AI

During the preparation of this work, the authors used the LLM models DeepSeek and Claude exclusively for translation (German to English) and for literature and research searches. The LLM models were not employed to generate core content of the paper, i.e., they were not involved in analysis, methodology, results, or conclusions. Additionally, the code discussed in the paper is written by the authors.

References

[1] A. Felfernig, L. Hotz, C. Bagley, J. Tiihonen, Knowledge-based Configuration – From Research to Business Cases, Morgan Kaufmann, 2014.

[2] L. Hotz, A. Felfernig, M. Stumptner, A. Ryabokon, C. Bagley, K. Wolter, Configuration Knowledge Representation & Reasoning, in: A. Felfernig, L. Hotz, C. Bagley, J. Tiihonen (Eds.), Knowledge-based Configuration – From Research to Business Cases, Morgan Kaufmann Publishers, 2014, pp. 59–96.

[3] J. Martin, Fourth Generation Languages, Volume I: Principles, Prentice-Hall, 1985.

[4] W. R. Harris, S. Gulwani, Spreadsheet table transformations from examples, in: Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, ACM, 2011, pp. 317–328.

[5] A. O. Shigarov, A. A. Mikhailov, Rule-based spreadsheet data transformation from arbitrary to relational tables, Information Systems 71 (2017) 123–136.

[6] K. David, Funktionales Kostensplitting zur Ermittlung von Mieterhöhungen nach energetischen Maßnahmen - Eine Handlungsempfehlung auf Basis theoretischer und empirischer Untersuchungen, doctoralthesis, HafenCity Universität Hamburg, 2019. URL: https://repos.hcu-hamburg.de/handle/hcu/505.

[7] G. L. J. Steele, COMMON LISP: the Language, Digital Press, 1984.

[8] J. M. Siskind, Screaming Yellow Zonkers, Technical Report, M.I.T. Artificial Intelligence Laboratory, 1991.

[9] Choco Team, PyChoco: Python binding for Choco-solver, https://github.com/chocoteam/pychoco, 2025.

[10] L. Hotz, K. Wilke, Structuring Legal Text as Preparation for Chat-Bot Use, in: Artificial Intelligence for Digital Public Services, STAF Workshop, CEUR Workshop Proceedings, 2025. To appear.