iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** V **Month of publication:** May 2026

DOI: https://doi.org/10.22214/ijraset.2026.82658

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue V May 2026- Available at www.ijraset.com

A Configurable Architecture for University Timetabling

Dhruv Kushwah[1]
[1]*Centre of AI, Madhav Institute of Technology and Science, Gwalior, Madhya Pradesh, India*

Abstract: ***Creating a university's timetable is a complex optimization problem with many constraints, which often leads to inefficient or error prone results. This paper proposes an automated end-to-end university timetable generation system that is configurable, and developed using Constraint Programming and the Google's CP-SAT Solver. The system allows users to define their constraints via a domain specific language, which facilitates a flexible means of creating diverse academic scenario models. The modular architecture separates the modeling of constraints from the optimization model, allowing non-expert users to create complex rules without needing to interact with the solver. Additionally, the system is implemented as an Asynchronous web-based back-end to support real-time handling of scheduling tasks. Its deployment in a live environment has demonstrated its effectiveness in scheduling conflict free time-tables for universities.***
Keywords: Constraint Programming, Timetabling, CP-SAT, Optimization, Resource Allocation.

I. INTRODUCTION

Creating a Timetable is traditionally very labor-intensive and depends on human involvement. Typically, our university administrators use unconnected spreadsheet software to create timetables, often manually matching several matrices. Not only is this an arduous task, but it has a high probability of miscalculating and creating conflicts in scheduling with both rooms and teachers. In the field of computer science, this has been mathematically established to be NP-hard. That is, as you increase the number of parameters (teachers, specific batches, and constraining rooms), you also significantly increase the number of possible combinations to complete the schedules. Therefore, it is impractical to perform manual trial and error or to use traditional brute-force programming to find the optimal solution from the massive number of combinations. Thus, to efficiently search through the entire set of possibilities will require an advanced mathematical solver.

This paper presents an interface that is supported by a strong computational pipeline. The system converts complicated institutional rules, like batch separations, multi-teacher assignments, and fixed time slots, into algebraic constraints by utilizing Google's CP-SAT solver. However, since the constraints are usually hard-coded into a software, it is not easy to define new complex rules, and even more so by the person who is not an expert in this field. To solve this problem, a simple domain specific language was implemented to enable clients to create new rules inside a user interface. The architecture also uses an asynchronous worker queue to separate the solver's heavy workload from the web interface, ensuring system stability and a smooth user experience.

II. RELATED WORK

In the field of scheduling and optimization, the university scheduling problem has been extensively researched [1]. The literature has presented a number of strategies to deal with its complexity. Heuristic and rule-based approaches, which are often easy to apply but have trouble handling large-scale and severely constrained scenarios, were the mainstay of early approaches.

For timetabling issues, metaheuristic techniques like Tabu search, simulated annealing, and genetic algorithms, great deluge algorithms have been thoroughly investigated in [2], [9]. These techniques seek to improve candidate schedules to find near-optimal solutions. Although they can yield good results, they may not be feasible for all constraints and frequently require careful parameter tuning.

Because constraint programming can explicitly model complex constraints, it has become a potent alternative to solve timetabling problems [3]. Constraint programming and integer programming techniques are combined in modern solvers, like Google's CP-SAT, to enable effective exploration of large solution spaces while guaranteeing constraint satisfaction [4].Hybrid approaches combining multiple constraints based optimization techniques have also been explored for timetabling problems [8].Other optimization techniques, such as mixed integer programming, have been studied to solve the problem with respect to various constraints and allocation of resources [10].

Recent literature has investigated the possibility of integrating these mathematical models into decision support systems and web-based API architecture in a way that directly assists administrative personnel [5],[7]. In addition, Literature has shown that CP frameworks greatly reduce the administrative burden of manual scheduling by effectively eliminating invalid timetable configurations [6]. In particular, using CP-SAT, has shown strong performance in real-world timetabling applications [11].
While some of the recent frameworks provide a functional web interface, they often function as static input/output portals, rather than dynamic tools for supporting decisions. Many of them lack real-time conflict detection, involve highly technical configuration steps, or are not documented as being used in a live environment.

III. SYSTEM ARCHITECTURE

To bridge the gap between mathematical complexity and actual usability, the proposed system is designed and implemented as a distributed full-stack web application. It is divided into 3 operational phases.

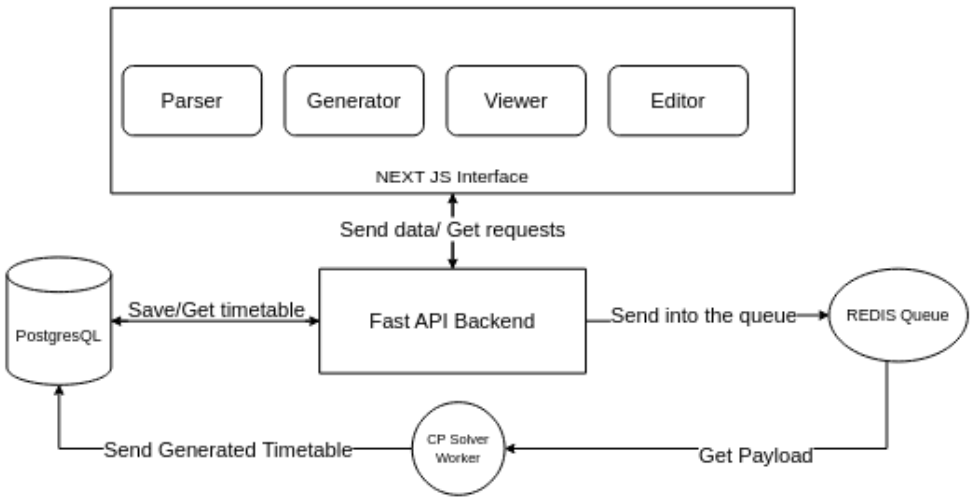


Fig.1 System Architecture

A. Data Ingestion

The user interface is built using the Next JS stack that offers a dashboard for data input. The users can feed the data through spread sheets. Users can even specify constraints in spread sheets, however, for more granular control, a simple Domain Specific Language was implemented, whose details are in Section IV. A five sheet format is used for data ingestion.

B. Asynchronous Processing Pipeline

After the user uploads the data, it is checked very carefully against a set schema to make sure that the solver does not fail. After the data has been checked, it is turned into a structured JSON payload and sent to the Backend. Because constraint programming is a computationally expensive problem, the system needs to make sure that the heavy math does not slow down the user's experience. To address this, an asynchronous worker pattern is implemented in the system.
When the payload has been received by the backend API, developed using Python FastAPI, it will be placed into a Redis queue immediately. A separate worker process will be polling the Redis Queue for new jobs. When a worker detects that a job exists, they will initialize the Google CP-SAT Solver, apply the mathematical formulation to the ingested data, and begin solving for the optimal schedule. Simultaneously, the Front-end will be actively polling the backend API for real-time status of the job that has been submitted to it.

C. Visualization, Storage and Editing

The final timetable matrix will be persisted in a PostgreSQL Database after the worker has successfully resolved and created the timetable. The front-end will then retrieve and display the information. Another component of the system is an interactive editor, which will allow a user to manually edit an algorithmically-generated timetable where there is an identified need for this to occur.

IV. FORMULATION OF CONSTRAINTS

Since the scheduling model is built using the CP-SAT solver, it is necessary to define constraints prior to generation. However, hard-coding constraints directly into the solver limits the flexibility of the model, and defining new constraints requires administrators to understand the underlying syntax of the program and recompile it every time. To solve this issue, a JSON-based Domain Specific Language (DSL) was engineered. This allows dynamic parsing of constraints into the Abstract Syntax Tree (AST) before being injected into the solver. But first, we need to formally define the constraints.

A. Decision Variables

In this scheduling model, the multidimensional boolean tensor is the main decision variable. Let B be the set of all branches, Sub_b be the set of all subjects for branch $b \in B$, and let D be the set of working days and T be the time slots where classes must be scheduled in a day. The decision variable can be represented as follows:

$$X_{b,s,d,t} \in 0,1$$

Hence, if $X_{b,s,d,t} = 1$, subject $s \in Sub_b$ is assigned on day d at time t; otherwise, $X_{b,s,d,t} = 0$. The system imposes strict hard constraints to ensure a valid timetable. Some of the core constraints and their formal representation are listed below :

B. Implemented Constraints

The system imposes strict hard constraints to ensure a valid timetable. Some of the core constraints and their formal representation are listed below :

1) Curriculum Fulfillment (Frequency Constraint):

Each subject must be scheduled according to their defined frequency F_s for subject s:

$$\sum_{d \in D} \sum_{t \in T} X_{b,s,d,t} = F_s \forall b \in B, \forall s \in Subject_b$$

2) Student Anti-Collision (Single Class per Slot):

Students cannot have overlapping schedule slots with regard to individual subjects (i.e., students cannot be scheduled for more than one subject at the same time).

$$\sum_{s \in b} X_{b,s,d,t} \leq 1 \forall b \in B, \forall d \in D, \forall t \in T$$

3) Faculty Anti-Collision:

Faculty members cannot be double-booked, nor can faculty members be in two different locations at the same time:

$$\sum_{b \in B} \sum_{s \in b \wedge f \in T \wedge acch(s)} X_{b,s,d,t} \leq 1 \forall f, \forall d, \forall t$$

4) Spatial Exclusivity (Room Anti-Collision):

To ensure the integrity of the physical infrastructure of the university, the system must prevent double-bookings of classrooms and laboratories. Let R be the set of all available rooms. Further let $V_{b,s,d,t,r} \in \{0,1\}$ where subject s from branch b is occupying room r at time t on day d, then the constraint is formally defined as:

$$\sum_{b \in B} \sum_{s \in b} V_{b,s,d,t,r} \leq 1 \forall r \in R, \forall d \in D, \forall t \in T$$

Additionally, an equality constraint ensures that if a subject is scheduled, it is allocated exactly one room:

$$\sum_{r \in R} V_{b,s,d,t,r} = X_{b,s,d,t} \forall b \in B, \forall s \in b, \forall d \in D, \forall t \in T$$

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue V May 2026- Available at www.ijraset.com

V. THE DOMAIN-SPECIFIC LANGUAGE

Having understood the structure of the core constraints, we can now create a flexible syntax to define new constraints. The syntax for defining a constraint using a domain-specific language can be validated using the Extended Backus-Naur Form (EBNF) to produce a formal grammar or Blueprints of Constraint. EBNF allows the mapping of iterative domain structures into a boolean expression tree that provides a dynamic way to perform higher-order aggregates and extract properties from the data held in that particular domain. As mentioned in the previous section, all constraints begin with the definition of the constraint against the data from which it is being created; therefore, our syntax has then followed that format and will start with the FORALL statement to define the set of data being used (the set of data is the first parameter), the WHERE statement to filter out the data from that set and finally by defining ASSERT to provide the definition of the expression.

1) FORALL (Scope): The iterative loops over the global datasets are created by defining a loop for each branch and all of its subjects (e.g., looping through all branches and subjects).

2) WHERE (Filter): A single binary expression tree containing logical operators (AND, OR) and relational operators which will dynamically filter the dataset. The system can then target a subset of data by applying a rule to only that data, for example, apply a rule only if the subject type is "online".

3) ASSERT (Action): The filtered multidimensional data are mapped into linear algebraic equations. Support is given for standard operators and also for any high order aggregates.

```
constraint    ::= "FORALL" iter_list
                  [ "WHERE" bool_expr ]
                  "ASSERT" expr

iter_list     ::= iter { "," iter }

iter          ::= id "IN"
                  ( string | property | list )

bool_expr     ::= expr
                  ( "AND" | "OR" | "IMPLIES" )
                  expr
                | "NOT" expr
                | expr
                ("=="|"!="|"<="|">="|"<"|">")
                expr
                | expr [ "NOT" ] "IN" expr

expr          ::= aggregation
                | target_call
                | math_expr
                | id
                | val

aggregation   ::= "SUM"
                  "("iter_list[ "WHERE" bool_expr ]")"
                  expr

target_call   ::= id "(" [ expr { "," expr } ] ")"

property      ::= id "." id
```

Fig. 2 EBNF Grammer

The system uses dynamic function injection and recursive positional unpacking to convert the human readable JSON block into strict integer programming parameters. The resulting grammar for the defined DSL is shown in Fig. 2.

VI. EXPERIMENTAL RESULTS

The performance of the system is evaluated on the basis of the computational latency associated with the Google CP-SAT worker process. The evaluation of the system is carried out using four datasets, which evaluates the complexity associated with the total number of branches and the total number of subjects. The performance is determined based on the time taken to obtain a feasible or optimal schedule, is recorded from the time the solver starts processing the payload in the asynchronous Redis queue until the schedule is successfully generated. The results are obtained by running the solver on a desktop system with 32 GB of ram and Intel i7 11800H octa-core processor.

TABLE 1
PERFORMANCE

B	Subjects	Rooms	Teachers	Solve Time	Status
3	33	14	11	0.8s	OPTIMAL
6	63	12	24	2.6s	FEASIBLE
9	100	12	26	11.49s	FEASIBLE
12	147	14	29	48.3s	FEASIBLE

From Table 1, it can be deduced that in a medium complexity scenario, which is defined by a total of 9 branches and 100 subjects, the system is able to attain a feasible schedule within 12 seconds, while in a highly complex scenario, which is defined by a total of 12 branches and 147 subjects, the system is able to attain a feasible schedule within 48 seconds, whereas it would take nearly a week for humans to manually schedule.

VII. CONCLUSION AND FUTURE WORK

The new scheduling system is a combination of theoretical combinatorial optimization and the practicalities of managing a university. It does this by integrating Google's CP-SAT solver into a distributed, asynchronous web architecture that allows it to automatically generate conflict-free schedules. The results of the experiment show a high degree of scalability for this architecture with medium-sized university instances that can produce optimal or strictly feasible schedules in seconds.

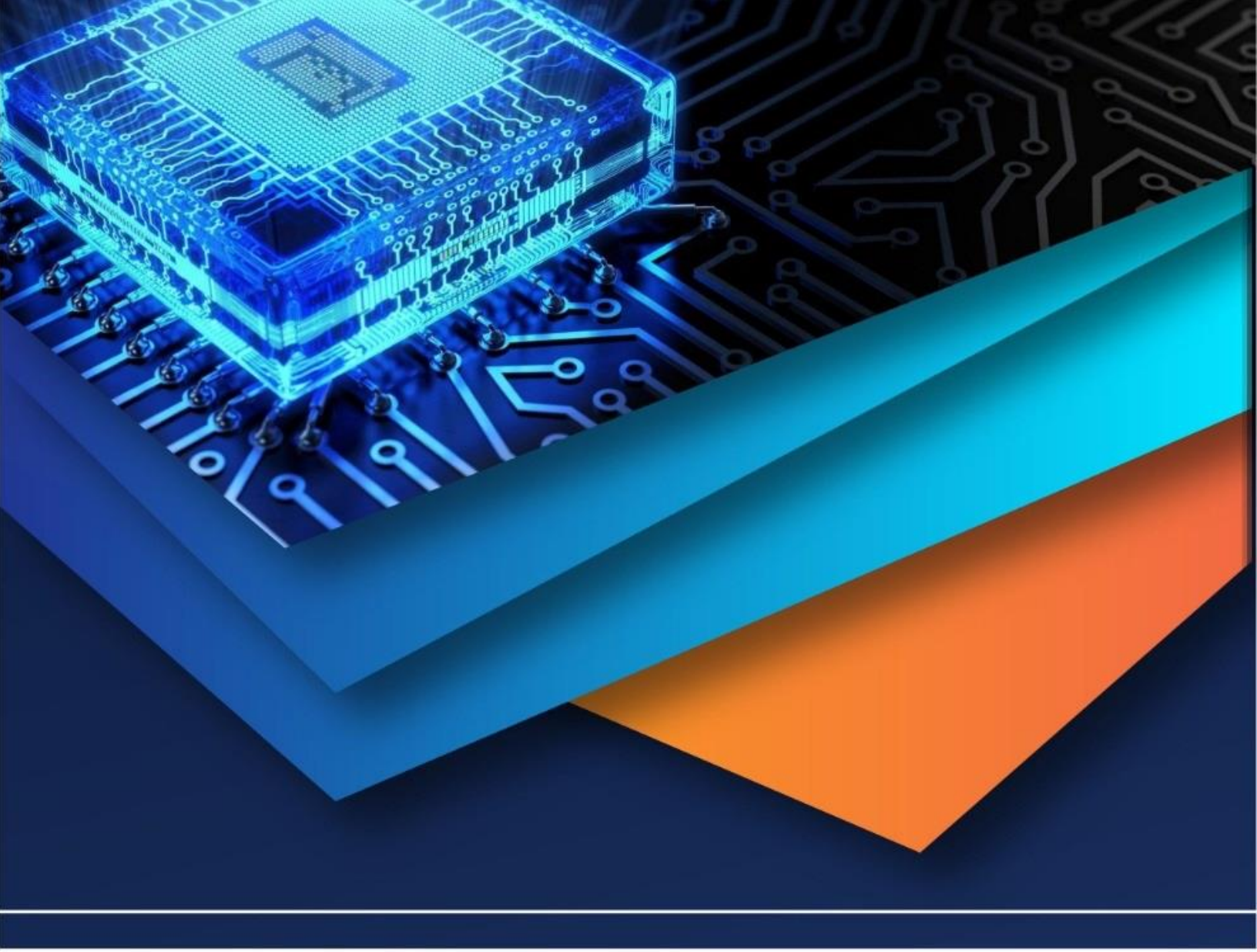
Creating a custom domain-specific language to separate the complex mathematical formulation from the user interface allows end-users to have considerable flexibility and dynamics in building constraints. The new DSL described here does not serve exclusively as a way to configure an interface; however the proposed solution builds on these capabilities by adding additional features and functionality such as allowing users to write constraints using quantified variables (FORALL), filter conditions based on those variables, aggregate data (SUM), and determine causal relationships between two objects (IMPLIES).

Future work will consider further democratizing the definition of constraints. The standard JSON format of the underlying DSL means that a Large Language Model could be trained to eliminate the need to define syntax rules manually and therefore allow administrative staff to completely configure highly-constrained timetabling models using conversational prompts.

REFERENCES

[1] E. K. Burke and S. Petrovic, “Recent research directions in automated timetabling,” European Journal of Operational Research, vol. 140, no. 2, pp. 266–280, 2002.
[2] M. W. Carter and G. Laporte, “Recent developments in practical course timetabling,” in Practice and Theory of Automated Timetabling II, E. Burke and M. Carter, Eds., Lecture Notes in Computer Science, vol. 1408. Berlin, Germany: Springer, 1998.
[3] A. Schaerf, “A survey of automated timetabling,” Artificial Intelligence Review, vol. 13, pp. 87–127, 1999.
[4] L. Perron and F. Didier, CP-SAT, version 9.15, Google, 2026. [Online]. Available: https://developers.google.com/optimization/cp/cp_solver. [Accessed: April 15, 2026].
[5] J. A. Garcia Yañez, F. J. Ramirez Saenz, E. L. Rios Saldaña, M. A. Rubio Aguilar, I. E. Barrios Flores, C. L. Garcia Flores, R. E. Hernandez Cerda, and J. J. Zacarias Ortiz, “Development of a web-based timetabling software for a Mexican university,” TAMIU Working Papers, 2024. [Online]. Available: https://www.tamiu.edu/cswht/documents/wp-2024-006-garcia-yanez.pdf.Accessed: Apr. 15, 2026.
[6] G. K. Keerthan, G. K. Praveen, C. M. Darshan, M. B. Nanda, and H. B. Sundeep Patil, “Automated timetable generation using constraint programming,” International Journal for Multidisciplinary Research (IJFMR), 2025.
[7] M. M. Alansari, “Optimized automatic course timetabling service architecture for integration with vendor management systems,” International Journal of Advanced Computer Science and Applications, pp. 878–890, 2022.
[8] T. Müller, “ITC2007 solver description: a hybrid approach,” Annals of Operations Research, vol. 172, pp. 429–446, 2009.

[9] S. Abdullah, H. Turabieh, B. McCollum, and P. McMullan, “A hybrid metaheuristic approach to the university course timetabling problem,” Journal of Heuristics, vol. 18, pp. 1–23, 2012.
[10] S. M. Al-Yakoob and H. D. Sherali, “Mixed-integer programming models for an employee scheduling problem with multiple shifts and work locations,” Annals of Operations Research, vol. 155, pp. 119–142, 2007.
[11] É. J. B. Moreira and S. A. A. De Freitas, "A CP-SAT Approach for Academic Resource Timetabling in Higher Education Institutions: A Case Study at a Major Public University," 2024 21st International Conference on Information Technology Based Higher Education and Training (ITHET), Paris, France, 2024, pp.1-8.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY