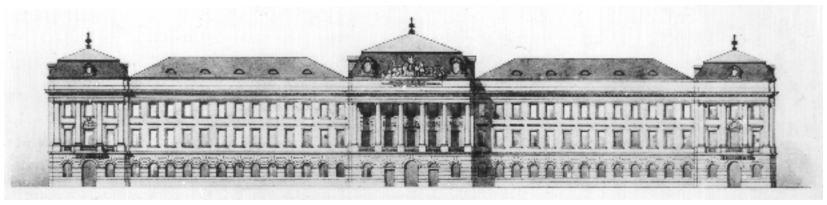
dbai

TECHNICAL REPORT



INSTITUT FÜR INFORMATIONSSYSTEME

ABTEILUNG DATENBANKEN UND ARTIFICIAL INTELLIGENCE

A General Modeling Format for Employee Scheduling

DBAI-TR-2017-109

Lucas Kletzander, Florian Mischek, Nysret Musliu, Gerhard Post and Felix Winter

Institut für Informationssysteme
Abteilung Datenbanken und
Artificial Intelligence
Technische Universität Wien
Favoritenstr. 9
A-1040 Vienna, Austria
Tel: +43-1-58801-18403
Fax: +43-1-58801-18493
sek@dbai.tuwien.ac.at
www.dbai.tuwien.ac.at

DBAI TECHNICAL REPORT
2017

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

DBAI TECHNICAL REPORT

DBAI TECHNICAL REPORT DBAI-TR-2017-109, 2017

A General Modeling Format for Employee Scheduling

Lucas Kletzander [1] **Florian Mischek** [1] **Nysret Musliu** [1]
Gerhard Post [2] **Felix Winter** [1]

Abstract. In this paper we present a structured XML format that is able to describe problem instances for a large number of different problems from the employee scheduling area. The format can be used to compose instances for new problems that combine specifications from several existing problems.

[1]TU Wien. E-mail: {lkletzan, fmischek, musliu, winter}@dbai.tuwien.ac.at
[2]University of Twente, E-mail: g.f.post@utwente.nl

Acknowledgements: This work has been supported by the Austrian Science Fund (FWF): P24814-N23.

Contents

1 Introduction

Employee scheduling problems appear in a wide range of fields including health care, airlines, transportation services and any other organization that employs a large workforce. A large number of different scheduling subproblems like Shift Design, Rostering, Break Scheduling and many others have been described and solved in the literature [Ernst *et al.*, 2004]. Due to the complexity inherent in this domain, these subproblems are usually solved separately, although there have been approaches combining two or more subproblems. Further, there are many variants of each such subproblem, as different industries and organizations have different requirements and constraints for their schedules. Most of the publications also introduce their own text based format to model their specific problem instances. As a result, solvers can not be reused for similar problem variants and comparison between different versions of the same problem are difficult, despite the often large similarities between them.

Especially for the subproblem of Employee Rostering, there have been some attempts at the categorization of problem variants (e.g. [De Causmaecker and Vanden Berghe, 2011]) and the development of a standardized instance description format. In particular, the XML based modeling format[1] for the AutoRoster software, developed by Staff Roster Solutions, supports a wide range of options and constraints. A set of benchmarking instances[2] from a large number of different publications and institutions has been modeled using this format.

The same company also provides a (separate) modeling format for a limited form of the shift design problem[3].

For other subproblems, and more complex problems, such a standardized format does not exist.

In this paper we present a structured XML format, which extends the AutoRoster modeling format, and is able to describe problem instances for a large number of different problems from the employee scheduling area. Additionally, the format can be used to compose instances for new problems that combine specifications from several existing problems. Using our format encourages the reuse of code when creating solvers and can also make it easier to compare the performance of different solution methods, since benchmark instances that use the format can easily be shared between publications.

In Section 2, we will give a description of the type of scheduling problems our format is designed to model. Section 3 will present the five core modules that make up our format. In Section 4 we will show how the format can be used to model some basic instances for existing shift design, break scheduling and rostering problems.

2 General Employee Scheduling

In 1986, Glover and McMillan [Glover and McMillan, 1986] gave an informal description of the *General Employee Scheduling* problem.

[1]`http://www.staffrostersolutions.com/support/autoroster-problem-data.php`
[2]`http://www.schedulingbenchmarks.org/`
[3]`http://www.staffrostersolutions.com/support/shiftsolver-problem-data.php`

Based on their model, with extensions to cover other important variants, the following elements can be identified in employee scheduling problems:

There is some form of *demand*, describing the work that needs to be done. Depending on the specific problem variant and industry, this is usually given either as a list of tasks that have to be performed or as staffing requirements during certain time periods.

This work is grouped into *shifts*, which denote consecutive periods of work for an employee. A large number of constraints govern the length, placement and structure of these shifts. In other problem variants, the set of feasible shift types is instead given in the problem description. These shifts can include *breaks* that have to be placed according to various rules.

Lastly, there is a pool of *employees* (of either fixed or variable size), which are available to actually perform the required work. These employees must be assigned to shifts according to many different constraints and regulations. Individual employees may differ according to their contracts, skills or working preferences.

A solution to the problem is a working schedule for each employee that fulfills all constraints or, in the case of optimization problems, minimizes the violations of soft constraints.

In most cases, not the full functionality offered by this formulation is needed.

- Most Shift Design and Shift Scheduling instances do not require the assignment of work to individual employees.
- Employee Rostering problems typically deal with a set of shift types that are fixed in time therefore do not require the construction of feasible shifts based on demands. In these cases it suffices to assign shifts one of a small set of shift types.
- The number of employees is often fixed and known in advance.
- Many settings do not require the scheduling of breaks.

All these reductions greatly simplify the structure of the problem and might enable different solution approaches.

3 Format Modules

The format that we present contains a large number of XML elements and attributes that can be used to describe the requirements and properties of many different scheduling problem instances, including all variants and subproblems described in the previous section. All of those attributes and elements are structured into six main modules, following the partition of problem elements identified above:

General contains global information about the scheduling horizon and the problem instance.

Tasks is an optional module defining all available task types that can be used in the solution.

Shifts defines all feasible shift types that can be used in the solution.

Breaks is an optional module defining the structure and placement of breaks.

Employees gives a list of available employees, either in the form of heterogeneous individuals or homogeneous abstract employee types.

Demands models the staffing requirements, either based on tasks, time slots or shift types.

In the following, the properties and options contained in each of those modules will shortly be described. A full documentation of each element can be found in [4].

3.1 General Module

The elements contained in the General module can be used to model general information about the problem instance.

This includes, among others, the *length* of the considered scheduling period, the *start*- and *end date* of the scheduling horizon and the length of the *shortest time interval* that should be considered for this instance.

Another set of elements contains flags for the solver concerning the task to be solved. This includes information about whether the schedule should be seen as *cyclical*, whether an *individual roster* should be generated for each employee and whether some sort of *shift design* is necessary.

3.2 Tasks Module

The optional Tasks module can be used to specify all task types that can appear in a feasible solution schedule.

A task type is a certain type of work that can be scheduled within certain time windows and requires at least one employee. This module can specify whether a task type counts as work time or free time, *prerequisite tasks* that need to be completed before this task type or the *re-acquaintance period* needed after performing this task type.

3.3 Shifts Module

Elements and attributes in the Shifts module can be used to specify all necessary information about the working shifts that can appear in a feasible solution schedule.

Different shift types can be defined by *starting* and *ending time* as well as *length*. These parameters can also be given as *ranges*, to allow the solver to determine concrete values for each instance of the shift type. Each block of work assigned to an employee must be assigned exactly one shift type and meet all its constraints.

Further, it is possible to limit the number of *shift instances* for each shift type, i.e. unique assignments, as defined by starting and ending time.

[4]`http://www.dbai.tuwien.ac.at/proj/arte/ges_format/`

3.4 Breaks Module

If the optional Breaks module is included, any requirements regarding breaks that should be scheduled during working shifts can be formulated.

This process is split into two main parts: First, different *break types* (e.g. ”lunch break”, ”short break”), are defined. These differ in their *length*, *position* within the shift, the *minimum* and *maximum* time spent *working before* and *after* the break and other criteria.

These break types can then be used in *break configurations*, which consist of a *filter* on the properties of shifts that should use this configuration and a definition of both the *number* and *order* of instances of each break type within the shift. In addition to that, it is also possible to set limits on the *total time* spent on break within the shift.

3.5 Employees Module

Information about the available workforce and feasible rosters can be defined within the Employees module.

Employees are split into two different types: Single *employees* and generic *employee groups*. The former represent heterogeneous individuals, while the latter are anonymous groups of homogeneous employees, of which a variable number can be hired. These types can also be mixed, in order to model e.g. a set of core employees and optional temporary hires to cover short term workloads.

Employees (of either type) can have *contracts* specifying the constraints that apply to their schedules (if multiple contracts are assigned to an employee, all constraints apply together). These contracts define the limits on the total *workload* of an employee, the *skills* provided, constraints on the *shifts* they can work as well as a wide range of restrictions on working *patterns*. These constraints can restrict sequences of *consecutive shifts*, working days or days off, *weekend work* as well as express other, more complex constraints. Supported is also a form of *fairness* between employees that limits the size of the gap between the highest and lowest workload assigned to any employee.

In addition to their contracts, employees can also have *preferences* to (not) work on certain days, shifts or tasks. Employees can also have *pre-assignments* that are shifts or periods of free time that must occur in any solution. These can also be given before the actual start of the scheduling horizon to include the previous time period for use in constraints.

The employee module also contains a description of all *skills* employees can have. Each skill may specify a set of other skills to subsume, although this can optionally incur a penalty.

Finally, employees can be constrained to (not) be *paired* up, such that that one employee works (or does not work) whenever the other does.

3.6 Demands Module

The Demands module can be used to state workforce demands that should be covered by a solution to the problem instance. The elements and attributes contained in this module provide options to formulate cover requirements in one of three different variants: *task cover*, *time cover* or *shift cover*.

Task cover demand is defined as a list of tasks that have to be allocated during the scheduling period. If more than one employee is needed for a task, it is assumed that all of them have to work on this task at the same time. Each task is of a certain task type and has a *length* and a *time window* during which it can be scheduled.

Time cover defines for each time slot the number of employees that should be present. If the *interval* given for an entry spans more than one time slot, then the staffing level must be met in each time slot during this interval.

Shift cover works similarly to time cover, except that demand is specified for each shift type instead. This is most useful in instances where the available shifts are predefined and no time-based information is present.

Independent of the variant used, multiple levels of both minimum and maximum cover *requirements* can be given, to model minimum and preferred staffing levels and similar situations. These requirements can also be differentiated by *skill* to require employees with a certain qualification.

4 Examples

In this section we describe how two scheduling problems that have been previously described in the literature can be modeled using our format. The given examples show how rostering, shift design and break scheduling problems can be formulated. The full source code of the examples is also available online[5].

4.1 Simple rostering example

The first example that we consider deals with a basic staff rostering problem based on the instances that have been described in [Curtois and Qu, 2014]. The goal for this problem is to construct an optimal staff roster for a fixed number of employees that fulfills a number of given hard and soft constraints.

Our simple example knows only one shift type of fixed length and the scheduling period is two weeks long. Figure 1 displays how the general information and shift type definition can be defined using our format.

4.1.1 Hard Constraints

The considered example defines seven different hard constraints. In the following we will describe each of them and explain how the constraints can be modeled using our format.

[5]`http://www.dbai.tuwien.ac.at/proj/arte/ges_format/examples/`

```
<SchedulingHorizon>
    <General>
        <StartDate>2017-01-02</StartDate>
        <EndDate>2017-01-15</EndDate>
     </General>
     <Shifts>
         <ShiftTypes>
             <ShiftType ID="D">
                <MinStartTime>09:00</MinStartTime>
                <MaxStartTime>09:00</MaxStartTime>
                <MinLength>480</MinLength>
                <MaxLength>480</MaxLength>
            </ShiftType>
        </ShiftTypes>
    </Shifts>
    [...]
</SchedulingHorizon>
```

Figure 1 This figure shows how the scheduling period and shift types are configured for our simple example.

Employees cannot be assigned more than one shift on a single day. This constraint is assumed to be a hard constraint for any rostering problem that is specified in our format and therefore does not have to be stated explicitly.

Minimum and maximum workload. This hard constraint restricts the total amount of working time that may be assigned to a single employee during the scheduling period. Figure 2 exemplifies how this constraint can be modeled with our format.

Minimum and maximum consecutive working days/days off. The example rostering problem considered in this section defines limits on the number of consecutive working days and consecutive days off, which can be easily stated using our format. Figure 3 displays a code example.

Maximum number of working weekends. An employee is considered to have a working weekend if he has to work either on Saturday, on Sunday, or on Saturday and Sunday. The hard constraint which limits the maximum number of working weekends can be stated using the formats *WeekendCount* element. Figure 4 displays how this constraint can be stated using our format.

Forced days off. In our example, some employees may state day off requests that are treated as hard constraints. Figure 5 shows how this can be modeled using fixed assignments in our format.

```
<SchedulingHorizon>
    [...]
    <Employees>
        <Contracts>
            <Contract ID="ContractA">
                <Workload>
                    <Max>72:00</Max>
                    <Min>56:00</Min>
                    <Unit>WorkTime</Unit>
                </Workload>
                [...]
            </Contract>
            [...]
        </Contracts>
        [...]
    </Employees>
    [...]
</SchedulingHorizon>
```

Figure 2 Workload requirements can be defined inside of contracts. The time limits defined in a contract then apply to all employees working under this contract.

4.1.2 Soft Constraints

Our simple rostering example defines different soft constraints that lead to a numeric penalty whenever they are violated. An optimal solution to this problem is any schedule that is feasible and leads to the overall lowest possible penalty.

In the following we will describe these soft constraints, and will explain how they can be modeled using our format.

Shift on/shift off requests. Employees can request to work in certain shift types on specific days in the schedule. Additionally, employees can also request not to work in certain shift types on specific days in the schedule. Figure 6 shows how such shift on/off requests can be modeled using our format. The given weights denote the penalty that will be given in case a request is violated.

Demands. The demands of our example define a preferred number of working employees per shift type for each day in the schedule. Different weights are declared for under- and over-coverage and the induced penalty is linearly dependent on the difference the number of assigned employees to the preferred number. Figure 7 shows how demands can be modeled using our format.

```
<SchedulingHorizon>
    [...]
    <Employees>
        <Contracts>
            <Contract ID="ContractA">
                <PatternConstraints>
                    <SequenceConstraint>
                        <Label>Min 2, max 5 consecutive shifts</Label>
                        <AnyShift/>
                        <Min>2</Min>
                        <Max>5</Max>
                    </SequenceConstraint>
                    <SequenceConstraint>
                        <Label>Min 2 consecutive days off</Label>
                        <NoShift/>
                        <Min>2</Min>
                    </SequenceConstraint>
                    [...]
                </PatternConstraints>
                [...]
            </Contract>
            [...]
        </Contracts>
        [...]
    </Employees>
    [...]
</SchedulingHorizon>
```

Figure 3 Inside of contracts, the *SequenceConstraint* elements can be used to specify any kind of consecutive sequence constraints. The wildcards *AnyShift* and *NoShift* can be used to match any shift type or a day off respectively.

4.2 Break scheduling example

In this section we describe a problem that includes shift design and break scheduling components and is based on the problem description from [Beer *et al.*, 2010]. The goal is to design shifts so that given time period based staff requirements are fulfilled. Additionally, breaks have to be scheduled during shifts without violating the workforce requirements.

In our second example we consider a problem instance that describes a cyclic scheduling period over one week. Four different shift types that have variable starting- and ending times are specified. All shifts can be scheduled as required as long as their variable starting times and lengths stay within the specified limits. However, in the final schedule there should be as few different instantiations per shift type as possible (Note that this does not restrict how often a certain shift type is

```
<SchedulingHorizon>
    [...]
    <Employees>
        <Contracts>
            <Contract ID="ContractA">
                <PatternConstraints>
                    [...]
                    <WeekendCount>
                        <Max>1</Max>
                    </WeekendCount>
                    [...]
                </PatternsConstraints>
                [...]
            </Contract>
            [...]
        </Contracts>
        [...]
    </Employees>
    [...]
</SchedulingHorizon>
```

Figure 4 The maximum weekends constraint can be realized using a pattern constraint. In this example the maximum number of working weekends is restricted to one.

scheduled, as long as all assignments have the same starting and ending time). Figure 8 shows how the shift type specifications and the general information for this example can be modeled using our format.

Since the goal of the second example is to design shifts and schedule breaks, the specification of individual employees is not necessary. In such a case we can specify an anonymous variable employee type in our format so that a solver program knows that it can use an arbitrary number of employees for the construction of the schedule. Figure 9 shows how such a variable employee type can be modeled using our format.

Our considered example specifies three different break types that should be scheduled during shifts: Short breaks, long breaks and lunch breaks. Shifts that are at least 360 minutes long require exactly one lunch break to be scheduled. Other breaks have to be scheduled whenever a certain amount of working time has passed. After a work period of 50 minutes or longer a long break has to be scheduled, otherwise a short break is sufficient. In our example it is required to schedule that many breaks, so that the total break time equals at least a quarter of the total shift length. Figures 10 and 11 show how the necessary specifications for this break scheduling problem can be modeled using our format.

Finally, the staffing requirements for our example are defined over specific time periods and can be easily modeled using *TimeCover* elements in our format. Figure 12 shows the use of those

```
<SchedulingHorizon>
    [...]
    <Employees>
        [...]
        <EmployeeList>
            <Employee ID="A">
                [...]
                <PreAssignments>
                    <NoShift>
                        <StartDay>0</StartDay>
                        <EndDay>0</EndDay>
                    </NoShift>
                </PreAssignments>
            </Employee>
            <Employee ID="B">
                [...]
                <PreAssignments>
                    <NoShift>
                        <StartDay>5</StartDay>
                        <EndDay>5</EndDay>
                    </NoShift>
                </PreAssignments>
            </Employee>
            [...]
        </EmployeeList>
    </Employees>
    [...]
</SchedulingHorizon>
```

Figure 5 *PreAssignments* can be used to define shift assignments on certain days that should not be changed in any way when generating a solution. In this example they are used to define strict day off requests.

elements in a code example.

References

[Beer *et al.*, 2010] Andreas Beer, Johannes Gärtner, Nysret Musliu, Werner Schafhauser, and Wolfgang Slany. An ai-based break-scheduling system for supervisory personnel. *IEEE Intelligent Systems*, 25(2):60–73, 2010.

[Curtois and Qu, 2014] Tim Curtois and Rong Qu. Computational results on new staff scheduling

```
<SchedulingHorizon>
    [...]
    <Employees>
        <EmployeeList>
            [...]
            <Employee ID="D">
                [...]
                <Preferences>
                    <ShiftOffRequest weight="3">
                        <Day>8</Day>
                        <ShiftTypes>D</ShiftTypes>
                    </ShiftOffRequest>
                    <ShiftOffRequest weight="3">
                        <Day>2</Day>
                        <ShiftTypes>D</ShiftTypes>
                    </ShiftOffRequest>
                    <ShiftOffRequest weight="3">
                        <Day>3</Day>
                        <ShiftTypes>D</ShiftTypes>
                    </ShiftOffRequest>
                    [...]
                </Preferences>
            </Employee>
            [...]
        </EmployeeList>
    </Employees>
    [...]
</SchedulingHorizon>
```

Figure 6 Shift requests can be modeled for each employee by providing the corresponding elements in their preferences.

benchmark instances. Technical report, ASAP Research Group, School of Computer Science, University of Nottingham, NG8 1BB, Nottingham, UK, October 2014.

[De Causmaecker and Vanden Berghe, 2011] Patrick De Causmaecker and Greet Vanden Berghe. A categorisation of nurse rostering problems. *Journal of Scheduling*, 14(1):3–16, 2011.

[Ernst *et al.*, 2004] A.T Ernst, H Jiang, M Krishnamoorthy, and D Sier. Staff scheduling and rostering: A review of applications, methods and models . *European Journal of Operational Research*, 153(1):3–27, 2004. Timetabling and Rostering.

[Glover and McMillan, 1986] Fred Glover and Claude McMillan. The general employee scheduling problem. an integration of ms and ai. *Computers & Operations Research*, 13(5):563 – 573,

```
<SchedulingHorizon>
    [...]
    <Demands>
        <ShiftCover>
            <Shift>D</Shift>
            <Day>0</Day>
            <Requirements>
                <Min weight="100">5</Min>
                <Max weight="1">5</Max>
            </Requirements>
        </ShiftCover>
        <ShiftCover>
            <Shift>D</Shift>
            <Day>1</Day>
            <Requirements>
                <Min weight="100">7</Min>
                <Max weight="1">7</Max>
            </Requirements>
        </ShiftCover>
        [...]
    </Demands>
</SchedulingHorizon>
```

Figure 7 This figure shows how demands per shift type can be defined in our format. Different weights are specified for under- and over-coverage by using separate *Min* and *Max* elements.

1986.

```
<SchedulingHorizon>
    <General>
        <PeriodLength>7</PeriodLength>
        <CyclicSchedule>Cyclic</CyclicSchedule>
        <TimeSlotLength>5</TimeSlotLength>
    </General>
    <Shifts>
        <ShiftTypes>
            <ShiftType ID="F">
                <Name>Early Shift</Name>
                <MinStartTime>05:30</MinStartTime>
                <MaxStartTime>08:00</MaxStartTime>
                <MinLength>360</MinLength>
                <MaxLength>765</MaxLength>
            </ShiftType>
            [...]
        </ShiftTypes>
        <GlobalShiftConstraints>
            <ShiftInstances>
                <ShiftTypes>F</ShiftTypes>
                <Max weight="60">1</Max>
            </ShiftInstances>
            [...]
        </GlobalShiftConstraints>
    </Shifts>
    [...]
</SchedulingHorizon>
```

Figure 8 This figure shows how the general information and shift type specifications for the second example can be modeled using our format. The timeslot length is specified to five minutes in this example. The code also shows how early shifts can be specified as a shift type. An early shift may start anytime between 5:30 and 8:00 and can last from 360 minutes up to 765 minutes. Preferably there should be only one instantiation of the early shift type and any additional instantiation will cause a penalty of 60.

```
<SchedulingHorizon>
    [...]
    <Employees>
        <Contracts>
            <Contract ID="C" />
        </Contracts>
        <EmployeeList>
            <VariableEmployee ID="E">
                <Contracts>
                    <Contract>C</Contract>
                </Contracts>
            </VariableEmployee>
        </EmployeeList>
    </Employees>
    [...]
</SchedulingHorizon>
```

Figure 9 This figure shows how an anonymous variable employee type can be modeled. The variable employee type E and contract C do not contain any preferences and just tell the solver that it can use an arbitrary number of employees when constructing a solution schedule for this problem. One could however use those elements to specify additional constraints if necessary.

```
<SchedulingHorizon>
    [...]
    <Breaks>
    <!-- Different break types are configured here, if no weight is given
        constraints are hard -->
        <BreakTypes>
            <BreakType ID="ShortBreak">
                <MinLength weight="1">10</MinLength>
                <MaxLength weight="1">15</MaxLength>
                <MinStartShift>30</MinStartShift>
                <MinEndShift>30</MinEndShift>
                <MinWorkBefore>30</MinWorkBefore>
                <MaxWorkBefore>45</MaxWorkBefore>
            </BreakType>
            <BreakType ID="LongBreak">
                [...]
            </BreakType>
            <BreakType ID="LunchBreak">
                [...]
            </BreakType>
        </BreakTypes>
        [...]
    </Breaks>
    [...]
</SchedulingHorizon>
```

Figure 10 This figure shows how break types can be modeled using our format. Different break types can be defined with *BreakType* elements. Different constraints regarding length, the starting and ending time relative to the shift, and the amount of working time that may appear between breaks can be formulated for each break type. In this example short breaks can have a length between 10 and 15 minutes and should not be scheduled within the first and last 30 minutes of a shift. Additionally, the length of a preceding working period should be between 30 and 45 minutes.

```
<SchedulingHorizon>
    [...]
    <Breaks>
        [...]
        <!-- Break configurations determine which breaks are allowed for
            which shifts -->
        <BreakConfigurations>
            <BreakConfiguration>
                <ShiftFilter>
                    <MinShiftLength>360</MinShiftLength>
                </ShiftFilter>
                <BreakSet>
                    <Break>ShortBreak</Break>
                    <Break>LongBreak</Break>
                    <Break MinCount="1" MaxCount="1">LunchBreak</Break>
                </BreakSet>
                <MinTotalBreakTimeFraction>0.25</MinTotalBreakTimeFraction>
            </BreakConfiguration>
            [...]
        </BreakConfigurations>
    </Breaks>
    [...]
</SchedulingHorizon>
```

Figure 11 *BreakConfiguration* elements can be used to state the break types that should appear in shifts that match given filter criteria. In this example, any shift that has a minimum length of 360 minutes can schedule all the three different break types, but has to assign exactly one lunch break. Furthermore, breaks should be scheduled in such a way so that the total break time sums up to 25 % of the shift's length.

```
<SchedulingHorizon>
    [...]
    <Demands>
        <TimeCover>
            <Day>0</Day>
            <Start>00:00</Start>
            <End>06:00</End>
            <Requirements>
                <Min weight="10">2</Min>
                <Max weight="2">2</Max>
            </Requirements>
        </TimeCover>
        <TimeCover>
            <Day>1</Day>
            <Start>00:00</Start>
            <End>06:00</End>
            <Requirements>
                <Min weight="10">2</Min>
                <Max weight="2">2</Max>
            </Requirements>
        </TimeCover>
        [...]
    </Demands>
</SchedulingHorizon>
```

Figure 12 This figure shows how time period based workforce requirements can be modeled using our format. In this example two employees should be present during 00:00 and 06:00 on the first two days of the schedule. The requirements are modeled as soft constraints, and therefore a violation will lead to a penalty that is weighted differently in case of an under- or over-coverage of the requirement.