Multi-Calendar Appointment Scheduling: Calendar Modeling and Constraint Reasoning

Stephanie Spranger and François Bry

Institute for Informatics, Univ. Munich, Germany
`http://www.pms.ifi.lmu.de`
contact: `spranger@pms.ifi.lmu.de`

Abstract. This article proposes a framework to model and reason on what we call *multi-calendar appointment scheduling problems*. Multi-calendar appointment scheduling problems such as planning a phone conference of three persons in France, Tunisia, and Japan refer to various temporal conditions, involving arbitrary calendric types such as months (Islamic months are defined differently from Gregorian months) possibly belonging to different calendars (e.g. the Gregorian calendar and the Islamic calendar). To solve such problems, we suggest a new class of Constraint Satisfaction Problems (CSPs); we call them Multi-Calendar CSPs. In this framework, the constraint variables are identified with finite, convex intervals in some calendric type and time constraints on such variables are expressed in terms of *typed* finite domain constraints. To solve time constraints which are defined on variables of different calendric types, a novel constraint, called *calendar conversion constraint* is proposed.

1 Introduction

This article proposes a framework to model and reason on *multi-calendar appointment scheduling problems* such as scheduling the phone conference of three persons in France, Tunisia, and Japan. The framework used to infer a possible time slot has to consider the personal and professional time constraints of the persons. Furthermore, everyone's calendar data preferably should be expressed in the calendar the person is used to and consider the different time zones involved. Problems such as multi-calendar appointment scheduling that involve arbitrary calendric data possibly referring to different calendars are, in particular, typical for the Web and the Semantic Web, since data on the Web is inherently distributed and heterogeneous. In fact, internationalization and localization are two important aspects the Web and Semantic Web community is striving for (`http://www.w3.org/International`). That means, modeling and reasoning on temporal and calendric data in the Web and Semantic Web, calendar data conversion, e.g. from Gregorian to Islamic months, from Gregorian weeks to Gregorian months or between Japanese and Gregorian year numbering that must be performed for calculations should be invisible to the users. Thus, solving such *multi-calendar appointment scheduling problems* should be aware of differences between cultural and/or professional calendars in use today.

So as to make it possible for every user to specify calendars and to express calendric data in the calendar he/she is used to, the Calendar and Time Type System CaTTS [1] has been developed. CaTTS consists of two languages, a *type definition language*, CaTTS-DL, and a *constraint language*, CaTTS-CL. CaTTS is based on a formal time model in accordance with the set-theoretic tradition of time granularity systems. However, CaTTS is purely interval-based, reflecting a widespread common-sense understanding of time: a time point such as "Tuesday, January 3 2006, 9 a.m.", expressed in the time granularity "hour" is internally modeled by a time interval with the duration of 1 hour.

This article is structured as follows. First, we briefly introduce into calendar modeling using the language CaTTS. Second, the basic principles and properties of Multi-Calendar Constraint Satisfaction Problems, Multi-Calendar CSPs for short, are introduced. Multi-Calendar CSPs refer to and rely on calendric types modeled in CaTTS. Such problems allow for modeling and solving multi-calendar appointment scheduling problems where appointments can be identified with finite, convex intervals in some calendric type and which are related by (metric) time constraints such as "A must start two days before B" or "an exam week must finish the running teaching term". Multi-Calendar CSPs are expressed in terms of *typed* finite domain constraints. Third, to solve constraints which involve variables of different calendric types, we suggest a *calendar conversion constraint* transforming the associated domains into some common calendric type. The calendar conversion constraint is based on an implicit coercion semantics for subtyping in CaTTS. Finally, we conclude and address some perspectives on multi-calendar appointment scheduling.

2 Principles of the Multi-Calendar CSP Framework

In [2] an approach to simple, i.e. point-based metric temporal reasoning with time granularities have been proposed. This framework allows for modeling and reasoning with simple temporal constraints on points (with time granularity) and distances between points (with time granularity) in a DLR Horn framework[1]. In this framework, one can express constraints like "at time t (in time granularity g), person A is in London", but neither "an event e (in time granularity g) happens during a task t (in time granularity h)" nor "an event e (in time granularity g) happens 5 time units (in granularity h) before an event e' (in time granularity k". To model and solve such metric temporal constraints, a more expressive framework is required; in particular, one must not only refer to time points but also to time intervals. Starting from the point-based model of [2], an interval can be represented by an ordered pair of points with the first point less than, i.e. before the second. To ensure that such intervals can meet, i.e. only having

[1] DLR (Disjunctive Linear Relations) [3] deals with the representation of temporal constraints by means of disjunctions of linear constraints (linear inequalities and inequations).

one endpoint in common, the endpoints i^- and i^+ of an interval must fulfill $i^- < i^+$. This can be achieved when the intervals are closed in their starting points and open on their ending points (or vice versa). However, this requirement is conflicting with the fact that a time model based on points does not correspond with an intuitive notion of time. Furthermore, with this point-based approach, the modeling of temporal knowledge is significantly complicated, because both, points and intervals have to be considered. Moreover, e.g. an event e holding sometime during an interval i which in turn is during an interval i' holds during i', as well. Thus, relations between intervals and events may be "carried forward" such that reasoning can be kept local. It is not clear how to maintain this locality when considering intervals modeled by endpoints in a point-based time model.

To overcome the problems of point-based time models, our approach to model and solve multi-calendar appointment scheduling problems is based purely on intervals. In practice, time points can be simulated by considering intervals that are small enough. For example, in scheduling a phone conference, time points can in general be presented by intervals with a duration of 1 minute or second, i.e. the smallest calendric type chosen. Representing time intervals using finite domains with domains typed after some calendric type, we are also able to handle quantifications such as "an event e (in time granularity g) happens 5 time units (in granularity h) before an event e' (in time granularity k". Furthermore, this interval-based approach opens us with the possibility to additionally incorporate generalized (i.e. finite, non-convex) and (infinite) periodic intervals, as well.

3 Multi-Calendar Appointment Scheduling

Multi-calendar appointment scheduling problems are a specific kind of metric temporal reasoning problems involving arbitrary calendric data and expressions. Let us consider the following appointment scheduling problem:

Example 1. A person plans a meeting lasting 3 working days after 20th April 2005 and before May 2005. A colleague's visit of 1 week must overlap with the planned meeting.

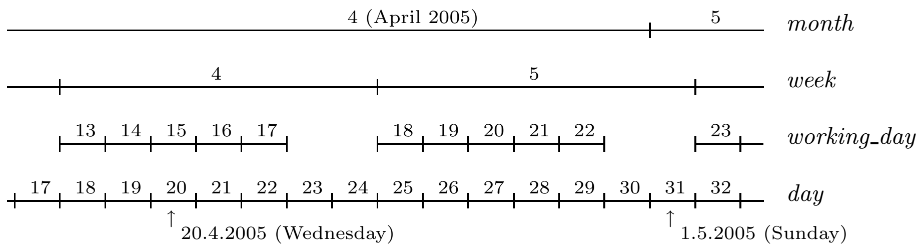


Fig. 1. The calendric types addressed in Example 1.

To properly analyze and solve such a problem, we are led to an abstract analyze of *activities* that take time, such as "meeting" and "visit". In our context, such activities can be identified with finite, convex intervals, isomorphic to subsets of the integers. Activities may refer to different *calendric types*. A calendric type is a (user-defined) calendar unit or a calendric expression such as "day", "working week", "teaching term", or "cs123 lecture"[2]. According to Example 1, the activity "meeting" refers to working days (i.e. Monday to Friday)[3] while the activity "visit" refers to weeks. Figure 1 illustrates the different calendric types addressed in Example 1. Activities are temporally related – either in terms of quantifications, e.g. the meeting must have a *duration of 3 working days* or in terms of interval relations, e.g. the meeting must be *before* 20th April 2005 or the visit must *overlap* the meeting. Such temporal relations are referred to as *time constraints*. The principle extension of our approach to (metric) temporal constraint satisfaction is that such time constraints may be applied to variables that refer to any user-defined calendric type. We call such CSPs *Multi-Calendar CSPs*.

The calendar type language CaTTS-DL [1] provides declarative language constructs to define calendric types such as those referred to in Example 1. In what follows, we illustrate how to specify a *CaTTS-DL calendar* that can be used to solve the afore introduced multi-calendar appointment scheduling problem:

```
calendar CalendarExample:Sig =
 cal
   type day;
   type week = aggregate 7 day @ day(-3);
   type month = aggregate
     31 day named january,
     alternate month(i)
      |(i div 12) mod 4 == 0 && (i div 12) mod 400 != 100 &&
       (i div 12) mod 400 != 200 && (i div 12) mod 400 != 300
        -> 29 day
      | otherwise -> 28 day
     end named february,
     31 day named march,
     30 day named april,
     31 day named may,
     30 day named june,
     31 day named july,
     31 day named august,
     30 day named september,
     31 day named october,
     30 day named november,
     31 day named december
    @ day(-90);
type working_day = select day(i) where
          relative i in week >= 1 && relative i in week <= 5;
end
```

[2] The Computer Science lecture with identifier 'cs123'.

[3] One might also refer to a calendric type "working day" that excludes weekend days and holidays, depending on some user's definition of such a type.

Each CaTTS-DL calendar specification has an identifier (in this example `CalendarExample`) that is assigned to a CaTTS-DL calendar specified within the keywords **cal** and **end**. `Sig` is the type of this calendar which is not given in this example.[4] The *reference* type of this calendar is `day`. It is specified by a parameterless type constructor. The type `week` is defined from the type `day` by using the CaTTS-DL **aggregate** type constructor. The type constructor's argument is a *predicate*. In case of type `week`, this predicate is defined over elements from type `day`. It is specified by a duration of 7 days (denoted `7 day`) and an anchor of this duration in days (denoted `day(-3)`). This type definition is read as follows: each element of type week has a duration of 7 days and weeks are anchored at days such that the first week, i.e. the week with index 1 starts with the day that has index -3. In this example, -3 refers to the Monday before 1st April 2005, i.e. 28th March 2005. Thus, the type `week` is defined in the CaTTS-DL calendar CalendarExample such that each week element starts on a Monday and has a duration of 7 consecutive days. The type `month` is also defined by a predicate over elements from the type `day` using the CaTTS-DL **aggregate** type constructor. This time, the type's predicate is specified by a periodic pattern with finite many exceptions, i.e. a finite sequence of durations (denoted `31 day` **named** `january, ..., 31 day` **named** `december`) and an anchor (denoted `day(-90)`). To define leap year regulations in Februaries, CaTTS-DL's **alternate** construct can be used within which the conditions of Gregorian leap year rules are formulated. The anchor is chosen such that the month with index 1 starts with the day with index -90. In this example, this means that January has index 1 and starts with 1st January 2005. Thus, April 2005 has the internal index 4, as illustrated in Figure 1. The type working_day is defined using the CaTTS-DL type constructor **select**. In this case, the type constructor has as argument a predicate over elements from type `day`, as well. The predicate **relative** `i` **in** `week` >= `1 &&` **relative** `i` **in** `week` <= `5` selects the first 5 days out of each week, that is Monday to Friday from each week.

The language CaTTS provides, in addition to the calendar type definition language CaTTS-DL, with the constraint language CaTTS-CL [1]. CaTTS-CL allows for modeling multi-calendar appointment scheduling problems which refer to calendars specified in CaTTS-DL. Note that CaTTS-DL also includes a formatting language CaTTS-FDL [4] in which formats for calendric types such as `"20.01.2006"` of type "day" and `"WS 2005/06"` (the winter term in years 2005/2006) of type "term". The problem illustrated in Example 1 can be modeled in CaTTS-CL as follows:

```
program SchedulingExample =
  prog
    use_calendar unqualified CalendarExample;
    use_format unqualified CatalogExample;

    Meeting is 3 working_day && Visit is 1 week &&
```

[4] The reader is referred to [4] for further readings on specifying and typing calendars using CaTTS-DL.

```
  Meeting after "20.04.2005" &&  Meeting before "05.2005" &&
  Visit overlaps Meeting
end
```

Each CaTTS-CL program has an identifier (here: SchedulingExample) that is assigned to a program specified within the keywords **prog** and **end**. The program initially imports one or more CaTTS-DL calendar specifications using the language construct **use_calendar** possibly followed by the keyword **unqualified**, i.e. using the imported calendar with short names. In the afore given CaTTS-CL program the CaTTS-DL calendar `CalendarExample` which has been specified in this section is imported unqualified. Furthermore, this CaTTS-CL program imports a CaTTS-FDL format catalog, denoted **use_format unqualified** CatalogExample. CaTTS-FDL is the format definition language of CaTTS with which one can define (date and time) formats for the elements of calendric types defined in some CaTTS-DL calendar specification such as "20.04.2005" of type `day` or "WS 2005/06" of type `teaching term`. A more detailed presentation of CaTTS-FDL can be found in [4]. For now and the rest of this article, we assume that a CaTTS-FDL catalog `CatalogExample` has been defined. The constraint variables for the activities `Meeting` and `Visit` are defined using the CaTTS-CL **is**-constraint followed by a duration, i.e. `Meeting` **is** `3 working_day` and `Visit` **is** `1 week`. The constraint `Meeting` **is** `3 working_day` is read as follows: The constraint variable Meeting is associated to the domain that consists of any interval of 3 consecutive working days, e.g. Friday, 22nd April 2005 - Monday, 25th April 2005 - Tuesday, 26th April 2005. `&&` denotes constraint conjunction. The conditions that the meeting must start after 20th April 2005, and that it must end before May 2005 are modeled using the CaTTS-CL time constraints **before** and **after** which have the common interpretations of the corresponding interval relations [5]. Similarly, the condition that the visit must overlap the meeting is modeled using the CaTTS-CL time constraint **overlaps**, denoted Visit **overlaps** Meeting which refers to the corresponding interval relation "overlaps"[5]. The complete CaTTS-CL syntax is given in Appendix A.

After having specified one or more CaTTS-DL calendars and some CaTTS-CL program that refers to some of those calendars, CaTTS-CL's *multi-calendar constraint solver* computes an *answer* to the specified problem. An answer to a CaTTS-CL program is a consistent Multi-Calendar CSP that is no further reducible. The answer to the problem given in Example 1 and modeled in CaTTS-CL (as previously illustrated) is the following:

```
Meeting is 3 working_day &&
(begin_of Meeting) within ["21.04.2005".."22.04.2005"] &&
Visit is 1 week &&
(begin_of Visit) within ["W04.2005".."W04.2005"]
```

The answer constrains the possible values the variables Meeting and `Visit` can be associated with: Meeting is a finite, convex interval of three consecutive working days (denoted `Meeting` **is** `3 working_day`) and the scheduled meeting can either start on the working day "21.04.2005" or on the working day

`"22.04.2005"`, denoted (**begin_of** `Meeting`) **within** `["21.04.2005".."22.04.2005"]`. The answer to the constraint variable `Visit` is given analogously.

Having the answer represented in CaTTS-CL, the programmer might ask for one or all *solution*s to this answer. A solution is computed by searching the reduced Multi-Calendar CSP using backtracking. The solutions to the considered problem are given in the following:[5]

```
(Meeting= ["21.04.2005".."25.04.2005"] && Visit= "W04.2005") ||
(Meeting= ["22.04.2005".."26.04.2005"] && Visit= "W04.2005")
```

The first solution `Meeting= ["21.04.2005".."25.04.2005"] && Visit= "W04.2005"` is read as follows: the constraint variable `Meeting` is assigned to the interval [”21.04.2005”..”25.04.2005”] from its associated domain (of working days) and the constraint variable `Visit` is assigned to the interval `"W04.2005"` from is associated domain (of weeks). The second solution is read analogously.

4 Multi-Calendar Constraint Satisfaction

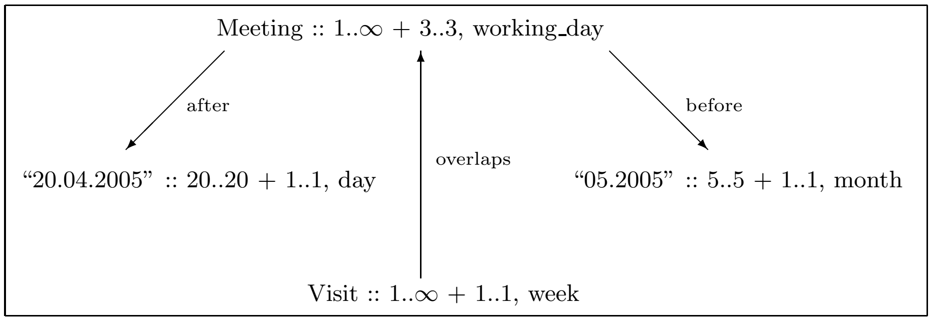

Meeting :: 1..∞ + 3..3, working_day
after
before
overlaps
“20.04.2005” :: 20..20 + 1..1, day
“05.2005” :: 5..5 + 1..1, month
Visit :: 1..∞ + 1..1, week

Fig. 2. Illustration of the appointment scheduling problem of Example 1 as constraint network.

A Multi-Calendar CSP is a finite sequence of variables with respective *typed* domains together with a finite set of *time constraints*, each on a subsequence of these variables. More formally, a time constraint is a subset of the Cartesian product of the domains of the variables. A time constraint is solved, if it equals this Cartesian product. An Multi-Calendar CSP is solved, if all its constraints are solved.

Multi-Calendar CSPs can be illustrated in terms of *constraint networks* by a directed graph where the variables (which represent activities) with their associated typed domains, called calendar domain expressions, are represented by nodes. The time constraints are represented by directed arcs between those variables according to Figure 2. The calendar domain expression ”Meeting ::

[5] || denotes a disjunction.

$1..\infty + 3..3$, working_day" represents the CaTTS-CL constraint `Meeting` **is** `3 working_day`. It is read as follows: the finite domain variable `Meeting` is associated to the domain $1..\infty + 3..3$, working day which consists of two ordinary finite domains, the first represents the meeting's starting time and the second one its duration. Those two domains are combined using an additive operation (denoted $+$).[6] "working_day" denotes the calendric type of `Meeting` which is defined in a CaTTS-DL calendar specification. I.e. the possible starting time of the meeting is any working day, represented by its internal integer index and the meeting has a duration of (minimal and maximal) 3 (working days). Possible values Meeting can be instantiated with are the intervals $[1,3]$, $[2,4]$, etc. The CaTTS-CL time constraint `Meeting` **after** `"20.04.2005"` is modeled by a directed arrow. The remaining time constraints and calendar domain expressions are modeled in the same manner.

Recall that a solution to a CSP is an assignment of each of the variables to values from their domains such that all the constraints in the CSP are satisfied. The idea to solve CSPs modeled as constraint networks is to remove those values from the domains of the variables that do not participate in a solution to the problem. That is, each time a time constraint like **after** is applied, the domains of the variables that participate in the constraint are reduced. But, if we try to apply the time constraint `Meeting` **after** `"20.04.2005"`, the following problem appears: `Meeting` is associated with a calendar domain of type `working_day` while `"20.04.2005"` is associated with a calendar domain of type `day`. Thus, using an ordinary finite domain solver, the Multi-Calendar CSPs illustrated in Figure 2 cannot be solved. Therefore, we suggest the following: solving Multi-Calendar CSPs involves constraint transformation and calendar domain conversion of some time constraint that is applied to variables of different calendric types. This solution is based on the principle of a coercion semantics for subtyping in CaTTS-CL together with a novel kind of constraint, called *calendar conversion constraint*. The conversion constraint relies on and refers to calendric types defined in CaTTS-DL: to apply any time constraint if the related variables have different calendric types, the calendar domains of the involved variables have to be converted into domains that represent the equivalent sets of times, however in some common calendric type both domains can be represented in. E.g. to apply the time constraint "overlaps" on the variables "visit" (of calendric type "week") and "meeting" (of calendric type "working day")", the associated calendar domains of "visit" and "meeting" have to be converted into equivalent domains represented in a calendric type "common" to types "week" and "working day" such as "day": a week is a set of 7 consecutive days and a working day is a specific day selected from the set of all days. Additionally, the representations of the associated calendar domains in the different calendric types are related to each other in terms of the conversion constraint. E.g. the conversion constraint relates the calendar domain representation of "visit" in

[6] Given the starting time s and the duration d of an interval, its ending time e is $e := s + d - 1$.

terms of type "week" to its calendar domain representation in terms of type "day".

4.1 The Constraint Solver

To solve Multi-Calendar CSPs, we have defined a *coercion semantics* for subtyping that compiles away subtyping during type checking a CaTTS-CL program against a CaTTS-DL calendar specification. In particular, this coercion semantics translates a CaTTS-CL program into an equivalent program in the constraint system *typedFD*. The constraint system typedFD is an extension of the constraint system Finite Domains [6] with (1) typed calendar domains and (2) the conversion constraint: $\forall \alpha, \beta \in \mathcal{C}.\ \alpha \leq \beta,\ X^\alpha \simeq Y^\beta$. α, β are calendric types defined in a CaTTS-DL calendar specification $\mathcal{C}$.[7] Using this constraint system typedFD, Multi-Calendar CSPs with finite domain variables can be solved. This extension takes advantage of existing local consistency algorithms approximating a solution to such problems which are known to be NP-hard. The main idea is to chose some subproblem in which to eliminate local inconsistency, and then iterate such elimination in all the chosen subproblems until stability. That means in particular that local consistency techniques make implicit inconsistencies explicit. To solve Multi-Calendar CSPs, we use bounds consistency [7], because the calendar domains are represented by a set of interval domains (i.e. one for the starting time and one for the duration).

The Underlying Coercion Semantics In principle, the coercion semantics is defined by a *transformation* of a CaTTS-CL time constraint $X^\gamma\ \mathcal{C}_t\ Y^\beta$ to a set of constraints in the system typedFD: $X^\alpha\ \mathcal{C}_t\ Y^\alpha \wedge X^\alpha \simeq X^\gamma \wedge Y^\alpha \simeq Y^\beta$. Since this transformation is done implicitly without the programmer guiding the (sub)type checker, it is called implicit coercion where subtyping is "compiled away" during type checking. The calendric type α is the *quasi-join*[8] of the calendric types γ and β defined in some CaTTS-DL calendar specification. The quasi-join is computed w.r.t. to the CaTTS-DL *subtype* relation, and it is denoted $\alpha := \gamma \vee \beta$.

Definition 1. (Subtype Relation) *Let τ and σ calendric types defined in CaTTS-DL. Then σ is a* **subtype of** *τ, denoted $\sigma \leq \tau$, if $\forall i \in \mathbb{Z}\ \exists j \in \mathbb{Z}.\ \tau(j) \subseteq \sigma(i)$.*

According to Example 1, *working_day* $\leq$ *day*, *week* $\leq$ *day*, and *month* $\leq$ *day*, but not *month* $\leq$ *week*.

Definition 2. (Calendar) *Let $\tau_1, \ldots, \tau_n$ be calendric types.*
A **calendar** *$\mathcal{C}$ is a finite set of calendric types $\{\tau_1, \ldots, \tau_n\}$ such that there exists a $\tau_i \in \mathcal{C}$ and for all $\tau_j \in \mathcal{C}$, $i, j \in \{1...n\}$ and $i \neq j$ $\tau_j \leq \tau_i$.*
$\tau_i \in \mathcal{C}$ is called **reference** *of the calendar $\mathcal{C}$.*

[7] X^α denotes a calendar domain expression of calendric type α where α is defined in some CaTTS-DL calendar specification $\mathcal{C}$.

[8] A quasi-join is slightly weaker than the ordinary lattice join.

According to Example 1, $\mathcal{C} = \{$*day, working_day, week, month*$\}$ where the reference type is *day*, i.e. *working_day* $\leq$ *day*, *week* $\leq$ *day*, and *month* $\leq$ *day.*

Note that a finite set $\mathcal{S}_{\mathcal{C}}$ of calendars is also a calendar according to Definition 2, if either the references of the calendars in $\mathcal{S}_{\mathcal{C}}$ are *aligned*, i.e. identical except for the numbering of their indices or there exists a type τ_0 which is $\leq$-comparable with the references of the calendars belonging to $\mathcal{S}_{\mathcal{C}}$. In particular, calendar alignment provides a means to reason with calendric types specified in different calendars, i.e. for multi-calendar reasoning. In the following, by $\mathcal{C}$ we refer to a finite set of aligned calendars.

To ensure calendric type conversion between any pair of calendric types in some calendar $\mathcal{C}$, we define a *quasi-join* of two calendric types according to the subtype relation. According to calendars, a quasi-join is slightly weaker than the ordinary lattice join, which only allows for the first condition of Proposition 1. The second condition of Proposition 1 is an extension to deal with calendars which are not (always) lattices. This exception results from the characteristics of the subtype relation, and, in particular, CaTTS-DL's possibility to construct a new calendric type by conjunction and/or disjunction of two other (already defined) types.

Proposition 1. *Let* $(\mathcal{C}, \leq)$ *be a calendar.*
For any pair of calendric types τ_t *and* τ_s *of* $\mathcal{C}$*, there exists a* **quasi-join** $\chi \in \mathcal{C}$ *such that* $\tau_t \vee \tau_s = \chi$*, i.e.* $\tau_t \leq \chi$*,* $\tau_s \leq \chi$*, and for all* $\tau_i \in \mathcal{C}$ *with* $\tau_s \leq \tau_i$ *and* $\tau_t \leq \tau_i$*, either*

1. $\chi \leq \tau_i$ *or*
2. $\tau_i < \chi$*, and there exists another* $\tau_k \in \mathcal{C}$ *with* $\tau_s, \tau_t \leq \tau_k$*,* $\tau_k < \chi$ *and* τ_i, τ_k *being incomparable.*

Thus, $(\mathcal{C}, \leq)$ is a *quasi-semi lattice.*

For example, the CaTTS-CL time constraint

$$\text{Visit}^{week} \text{ overlaps } \text{Meeting}^{working\_day}$$

is transformed as follows into constraints in the system typedFD: computing the quasi-join of the CaTTS-DL types `week` and `working_day` (according to the CaTTS-DL calendar specified in Section 3) yields the type `day`. Recall that in this CaTTS-DL calendar specification, types `week` and `working_day` has been both specified as subtypes from type `day` by predicates p_{day} and q_{day} aggregating (resp. selecting) subsets from the set of days:

$$\begin{tikzcd}
 & \text{week (aggregate p}_{day}\text{)} \\
\text{day} \arrow[ur, dash, "\geq"] \arrow[dr, dash, "\geq"'] & \\
 & \text{working\_day (select q}_{day}\text{)}
\end{tikzcd}$$

The following typedFD constraint then replaces the original CaTTS-CL time constraint:

$$\text{Visit}^{day} \text{ overlaps Meeting}^{day} \wedge$$
$$\text{Visit}^{day} \simeq \text{Visit}^{week} \wedge$$
$$\text{Meeting}^{day} \simeq \text{Meeting}^{working\_day}$$

That is, the time constraint "overlaps" is now applied on calendar domains of "Visit" and "Meeting" represented in the quasi-join type "day" of the original types "week" (resp. "working_day) of "Visit" (resp. "Meeting"), and the conversion constraints relates those two pairs of different representations of time in different calendric types to each other. Thus, using the conversion constraint, the temporal information hidden within the different calendric types is preserved during constraint solving Multi-Calendar CSPs.

The Calendar Conversion Constraint The core of the multi-calendar constraint solver for Multi-Calendar CSPs is its so-called *conversion constraint*:

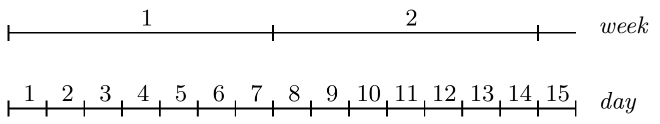
$$\forall \alpha, \beta \in \mathcal{C}.\ \alpha \leq \beta,\ X^{\alpha} \simeq Y^{\beta}$$

where α, β are calendric types defined in a CaTTS-DL calendar specification $\mathcal{C}$.

In principle, the conversion constraint defines a kind of "equivalence relation" between calendar domains that refer to different calendric types. It relies on and refers to calendric types defined in some CaTTS-DL calendar specification. In particular, for each pair of calendric types α and β where $\alpha \leq \beta$ in a CaTTS-DL calendar specification, a *conversion function* [4] is automatically generated from the (user-defined) type predicates by CaTTS-DL's language processor *before* performing any constraint propagation. Those functions are applied during constraint solving whenever a conversion constraint is propagated on variables of types α and β. Applying the conversion constraint on X^{α} and Y^{β}, new calendar domain expressions are added for X and Y. The new domains are such that the possible time intervals of X in type α correspond to the possible time intervals of Y in type β, and vice versa. This is achieved by adapting the bounds of the starting times of X and Y such that the bounds of X (which refer to type α) are started by the bounds of Y (which refer to type β) and by adapting the bounds of the durations of X and Y such that the bounds of X (which refer to durations of type α) correspond to the bounds of Y (which refer to durations of type β). The conversion constraint thus ensures that the bounds of the interval domains that represent the starting times of X in type α are equivalent to the bounds of the interval domains that represent the starting times of Y in type β. Furthermore, this constraint ensures that the bounds of the intervals that represent the durations of X in type α always correspond to the bounds of the intervals domains that represent the durations of Y. For example,

$$X :: 1..8 + 7..7, day \simeq Y :: 1..2 + 1..1, week$$

according to the following illustration:



Then the time interval $[8, 14]$ of days represented by X corresponds to the time interval $[2, 2]$ of weeks represented by Y, for example.

Thanks to the conversion constraint, (transformed) time constraints such as Visit^{day} overlaps Meeting^{day} can be propagated using an ordinary finite domain solver over integers.

4.2 Properties of the Constraint Solver

Now let us turn our attention to some formal properties of the multi-calendar constraint solver. We present those properties in short without giving the complete proofs here, just highlighting its basic ideas.

Completeness The proposed constraint solver for Multi-Calendar CSPs is *complete*, i.e. if the solver terminates with an equivalent bounds consistent problem, then the original problem is (globally) consistent. This can be shown in two steps:

1. solution generation from a bounds consistent Multi-Calendar CSP is straightforward, and
2. if the Multi-Calendar CSP is consistent, then the algorithm terminates with bounds consistency.

Concerning the first claim, it is first to note that all time constraints are represented in terms of classical finite domain constraints for which completeness has been shown [7,8]. Thus, it remains to show that from a bounds consistent conversion constraint a solution can be computed straightforward from the domains the participating variables are associated with. This is shown by straightforwardly applying the conversion constraint $\forall \alpha, \beta \in \mathcal{C}.\ \alpha \leq \beta,\ X^{\alpha} \simeq Y^{\beta}$ on the minimal solutions for X^{α} and Y^{β} which are always solutions to the considered Multi-Calendar CSP.

Concerning the second claim, the argumentation for termination of classical CSPs with finite domain variables also applies for the conversion constraint. As with classical CSPs, the algorithm for Multi-Calendar CSPs terminates with failure, if the original Multi-Calendar CSPs is inconsistent [4].

Complexity Testing for consistency of a Multi-Calendar CSPs in the afore introduced framework is *linear* in both the number of constraints and the number of variables w.r.t. the size of the calendar domains. This follows from the argumentation that testing consistency of classical CSPs over finite domain variables where the domains are represented by intervals is linear [8,6] and from the fact that the access to a CaTTS-DL conversion function in the conversion constraint can be done in *constant time*. Thus, the algorithm remains linear for Multi-Calendar CSPs with finite domain variables. The conversion function access can be done in constant time, since the conversion functions can be always generated from the types' predicates *before* performing any constraint propagation, at least by approximating the conversion function [4].

Note that to search for one or all solutions to a bounds consistent Multi-Calendar CSPs is NP-hard.

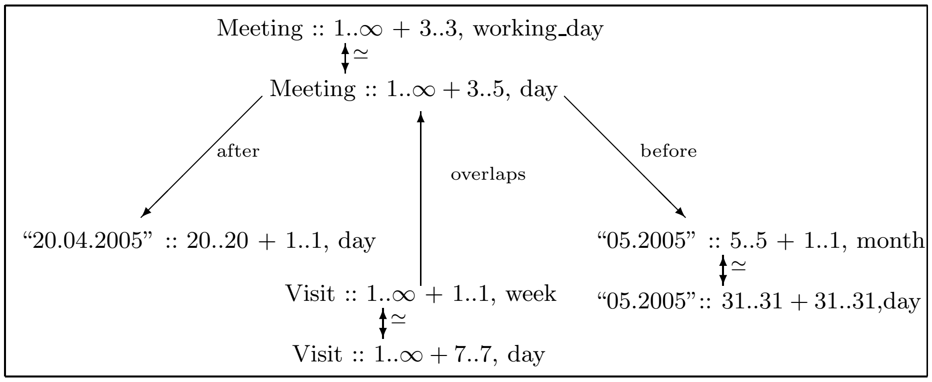

Meeting :: 1..∞ + 3..3, working_day
≃
Meeting :: 1..∞ + 3..5, day
after
overlaps
before
"20.04.2005" :: 20..20 + 1..1, day
"05.2005" :: 5..5 + 1..1, month
≃
"05.2005":: 31..31 + 31..31,day
Visit :: 1..∞ + 1..1, week
≃
Visit :: 1..∞ + 7..7, day

Fig. 3. Illustration of the appointment scheduling problem of Example 1 with conversion constraints as constraint network.

4.3 Example

Let us turn our attention back to Example 1. Having transformed the CaTTS-CL program given in Section 3 into the constraint system typedFD as previously described, this problem can be now illustrated by the constraint network given in Figure 3. This example is used in the following to demonstrate the working of the multi-calendar constraint solver.

Now we can reduce the calendar domains in the calendar domain expressions in this constraint network by applying the different time constraints which always involve applications of the related conversion constraints: applying the time constraint "after" on "Meeting" and "20.04.2005", reduces the domain of "Meeting" such that its smallest possible starting time is the day "21.04.2005", i.e. Meeting :: $21..\infty + 3..5$, day[9], and, therewith, Meeting :: $16..\infty + 3..3$, working_day according to the indexing used in Figure 1 which is computed by propagating the conversion constraint on the two different calendar domains of the variable "Meeting". That is, we have removed all those days possibly starting the interval "Meeting" that do not satisfy the constraint "Meeting after 20.04.2005". We say, that the constraint "after" propagates to the Meeting that it cannot start before "21.04.2005" which, in turn propagates this information into an equivalent set expressed in terms of working days. In the same way, we proceed by applying the remaining time constraints (and thus the associated conversion constraints)

[9] Note that the variable duration of this constraint (denoted 3..5) results from the conversion: three consecutive working days might be Friday-Monday-Tuesday. In terms of days this is Friday-Saturday-Sunday-Monday-Tuesday, i.e. 5 days.

until the Multi-Calendar CSP is not further reducible, i.e. until no further constraint propagation would reduce any of the considered domains. The not further reducible Multi-Calendar CSP is illustrated in Figure 4.

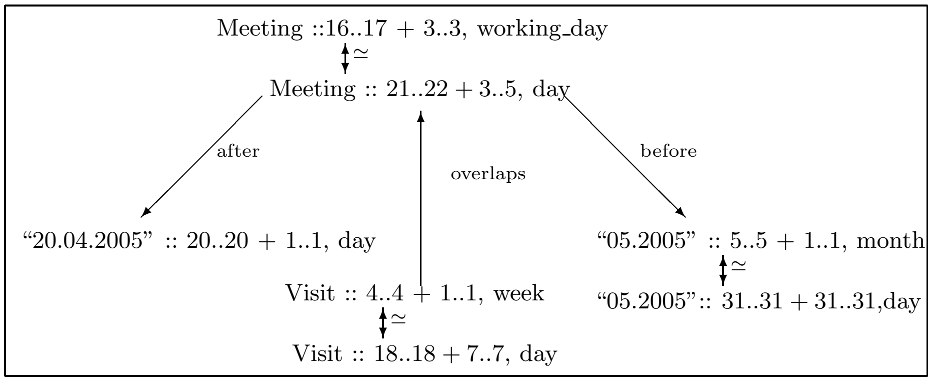

Meeting ::16..17 + 3..3, working_day
≃
Meeting :: 21..22 + 3..5, day
after
overlaps
before
"20.04.2005" :: 20..20 + 1..1, day
"05.2005" :: 5..5 + 1..1, month
≃
"05.2005":: 31..31 + 31..31,day
Visit :: 4..4 + 1..1, week
≃
Visit :: 18..18 + 7..7, day

Fig. 4. Illustration of the answer to the appointment scheduling problem of Example 1 as constraint network.

5 Related Work

We have proposed a framework to model and to reason on multi-calendar appointment scheduling problems. Calendar modeling is done using the language CaTTS-DL. Multi-Calendar CSPs can be modeled in CaTTS-CL. Such CSPs rely on and refer to calendric types defined in CaTTS-DL. In the framework of CaTTS, the notion of *calendric type* is based on the concept of *time granularities* [9,10,2,11]. Time granularities are well-investigated in theory in both Artificial Intelligence and Database Systems. In practice, time granularities are (at least to some extend) implemented with TSQL, however, not widely applied in temporal and active database systems. A novelty of CaTTS is however that CaTTS is a programming language approach defining time granularities as types. Furthermore, CaTTS introduces a novel relation between time granularities, i.e. the subtype relation. Note that CaTTS' subtype relation is derived from two more restrictive relations *aggregation of* and *inclusion of* [1] sufficient to specify almost any calendric type in use today, modifying relations between time granularities out of the large set of (possible) relationships between time granularities [2].

Multi-Calendar constraint satisfaction as proposed involves metric temporal constraints over intervals. Multi-Calendar CSPs go beyond simple, metric temporal reasoning with time granularities [2]. In particular, using merely simple, metric temporal reasoning frameworks (which provide with constraints to model points and distances between points in a DLR Horn framework) one could express temporal constraints such as “person A is at the bank”, but neither “an event e happens during a task t” nor “an event e happens 5 time units before an event e”’.[10] Such constraints can be however expressed in the proposed Multi-Calendar CSP framework where such CSPs are modeled by metric, interval-based time constraints in a typed finite domain constraint framework which allows for reasoning on such constraints where the domains of the variables possibly refer to different calendric types.. Furthermore, the suggested framework for Multi-Calendar CSPs yields a complete solution to the problem of time granularity conversion addressed in [12]. Actually, time granularity conversion is inherent to the operational semantics of the language CaTTS-CL.

Multi-Calendar CSPs inherently differ form qualitative temporal constraint problems [13,5,14] where (temporal) relations such as interval relations are specified and propagated between objects. An example of a merely qualitative temporal reasoning problem is the following: different persons come and leave a bank under certain constraints which are expressed in terms of temporal relations. Then one might ask questions such as “Could possibly persons A and B *meet* at the bank’s entrance?”. We claim that such kinds of questions does not frequently appear in appointment scheduling problems considered so far. Qualitative temporal reasoning thus performs reasoning on the temporal relations that hold between the different temporal objects whereas multi-calendar constraint satisfaction performs reasoning on the temporal objects (which refer to intervals with calendric types) themselves.

To summarize, Multi-Calendar CSPs are intended to model and solve (simple) multi-calendar scheduling problems where activities can be identified with finite convex intervals in any calendric type. Conditions on and between such activities are restricted to metric time constraints which can be modeled in terms of typed finite domain constraints. Furthermore, our approach provides with a complete and elegant solution to calendric type conversion for multi-calendar constraint satisfaction. The proposed constraint solver goes beyond proposals for simple, i.e. point-based metric temporal reasoning with time granularities.

Note that the proposed constraint reasoner is not designed for solving optimization problems like “schedule working shifts of workers including the worker’s specified preferences”. Such problems might involve soft constraints [15] and/or preferences [16], not expressible using CaTTS and not solvable using the multi-calendar constraint solver proposed in this article.

6 Perspectives

This article has introduced a framework for Multi-Calendar CSPs, in particular, to model and solve multi-calendar appointment scheduling problems. Multi-Calendar CSPs base on the calendar modeling language CaTTS. Multi-Calendar

[10] Such constraints cannot be expressed in qualitative temporal reasoning frameworks, as well.

CSPs are a new class of CSPs over finite domain variables with typed finite domain variables along with a novel constraint, the conversion constraint for calendar domain conversion of different calendric types defined in a CaTTS-DL calendar specification. Multi-Calendar CSPs have the same good properties as classical CSPs, i.e. consistency test is linear and the solver is complete.

As already mentioned, Multi-Calendar CSPs only support constraint variables which can be represented by finite convex intervals in some calendric types. On the one hand, this restriction to finite intervals over a calendric type allows for an efficient implementation of a constraint solver. On the other hand, this restriction considerably limits the set of temporal reasoning problems that can be modeled and solved: several problems such as travel planning involve calendric data that refers to generalized, i.e. not necessarily convex and finite intervals. To model and solve such problems, the time constraints provided with CaTTS-CL need to be extended such that they can propagate variables that refer to generalized intervals. In addition, time constraints (e.g. particular relations that often appear between non-convex time intervals, or time constraints used in order to relate periodic intervals such as Tuesdays are "periodically after" Mondays) will be then useful.

Another extension concerns the specification of preferences on CaTTS-CL time constraints. Furthermore, to model multi-calendar optimization problems, an extension of the introduced Multi-Calendar CSP framework with soft constraints might be useful.

Acknowledgment

This research has been funded in part by the PhD Program Logics in Computer Science (GKLI) and the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Program project REWERSE number 506779 (cf. `http://rewerse.net`).

The authors thank Frank André Rieß who has worked on the implementation of the multi-calendar constraint solver and Arnaud Lallouet for valuable and fruitful discussions.

References

1. Bry, F., Rieß, F.A., Spranger, S.: CaTTS: Calendar Types and Constraints for Web Applications. In: Proc. 14^{th} Int. World Wide Web Conference, Japan. (2005)
2. Bettini, C., Jajodia, S., Wang, S.: Time Granularities in Databases, Data Mining, and Temporal Reasoning. Springer-Verlag (2000)
3. Jonsson, P., Bäckström, C.: A unifying approach to temporal constraint reasoning. Artificial Intelligence **102** (1998) 143–155
4. Spranger, S.: Calendars as Types – Data Modeling, Constraint Reasoning, and Type Checking with Calendars. PhD Thesis. Herbert Utz Verlag, München (2006)
5. Allen, J.: Maintaining Knowledge about Temporal Intervals. Communications of the ACM **26** (1983) 832–843

6. Frühwirth, T., Abdennadher, S.: Essentials of Constraint Programming. Cognitive Technologies. Springer-Verlag (2003)
7. van Hentenryck, P., Saraswat, V., Deville, Y.: Constraint Processing in cc(FD). Technical Report, unpublished Manuscript (1992)
8. Tsang, E.: Foundations of Constraint Satisfaction. Academic Press (1993)
9. Montanari, A.: Metric and Layered Temporal Logics for Time Granularity. ILLC Dissertation Series 1996-02, University of Amsterdam (1996)
10. Jensen, C., (eds.), C.D.: The consensus glossary of temporal database concepts - February 1998 version. (1998)
11. Euzenat, J.: Granularity in Relational Formalisms with Applications to Time and Space Representation. Computational Intelligence **17** (2001) 703–737
12. Franceschet, M., Montanari, A.: A Combined Approach to Temporal Logics for Time Granularity. In: Workshop on Methods for Modalities. (2001)
13. Vilain, M.: A System for Reasoning about Time. In: Proceedings of the 2nd National (US) Conference on Artificial Intelligence, AAAI Press (1982) 197–201
14. Meiri, I.: Combining Qualitative and Quantitative Constraints in Temporal Reasoning. Artificial Intelligence **87** (1996) 343–385
15. Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based Constraint Satisfaction and Optimization. Journal of the ACM **44** (1997) 201–236
16. Prestwich, S., Rossi, F., K.B.Venable, Walsh, T.: Constrained CP-Nets. In: Italian Conference on Computational Logic. (2004)

A The Syntax of CaTTS-CL

ce ::=	*CaTTS-CL expressions:*
k	*constant*
X	*variable*
i	*index*
n τ	*duration,* $n \in \mathbb{N}$
[ce..ce]	*endpoint interval*
ce upto ce	*duration interval*
ce downto ce	*duration interval*
X is 1 τ	*event*
X is τ	*task*
X is n τ	*task with duration* $n \in \mathbb{N}$
ce $\mathcal{R}$ ce	*interval constraint*
ce $\odot$ ce	*metric constraint*
duration ce	*duration of*
index ce	*index of*
begin_of ce	*begin of*
end_of ce	*end of*
shift ce forward ce	*forward shift*
shift ce backward ce	*backward shift*
extend ce by ce	*interval extension*
shorten ce by ce	*interval shortening*
relative ce in τ $\langle \odot$ i $\rangle$	*relative in*
relative ce to τ $\langle \odot$ i $\rangle$	*relative to*
min(ce)	*minimum*
max(ce)	*maximum*
ce && ce	*conjunction*

where

$\mathcal{R} \in$ {equals, before, after, starts, started_by, finishes, finished_by, during, contains, meets, met_by, overlaps, overlapped_by, within, on_or_before, on_or_after}
$\odot \in \{==, <=, <, >, >=, !=\}$
τ calendric type defined in a CaTTS-DL calendar specification

B Proof

Proof. **(Proposition 1)** For a pair of types τ and σ of calendar $\mathcal{C}$, consider the set of upper bounds $U(\tau, \sigma) = \{\upsilon | \tau \leq \upsilon, \sigma \leq \upsilon\}$.
(Existence) If $\tau_s = \tau_t = \alpha$, with α being the top element of C, then $U(\tau_s, \tau_t) = \{\alpha\}$, and our proposition is satisfied through *(1)*. So, if $\tau_s \vee \tau_t$ exists, so does $\tau'_s \vee \tau_t$, with τ'_s direct subtype of $\mathcal{C}_s$: If $\tau_s \leq \tau_t$ so is $\tau'_s \leq \tau_t$ and in this case $\mathcal{C}_t$ is the join, as $\tau_t \in U(\tau'_s, \tau_t)$ satisfies *(1)*. In case of $\tau_t < \tau_s$, either $\tau_t \leq \tau'_s$ and thus $\tau'_s \in U(\tau'_s, \tau_t)$ satisfies *(1)*, or else τ_t and τ'_s are incomparable and thus $\tau_s \in U(\tau'_s, \tau_t)$ satisfies *(1)*. Finally, if $\mathcal{C}_t$ and τ_s are incomparable, τ'_s cannot be greater than or equal to τ_t, because then τ_t would have to be less than of equal τ_s; either τ'_s, too, is incomparable to $\mathcal{C}_t$ and thus $\tau'_s \vee \tau_t = \tau_s \vee \tau_t \in U(\tau'_s, \tau_t)$ satisfies *(2)*, or else $\tau'_s \leq \tau_t$ and thus $\tau_t \in U(\tau'_s, \tau_t)$ satisfies *(1)*.
(Uniqueness) Be $\chi = \tau_s \vee \tau_t$. Let's assume χ' would also qualify as a join of τ_s and τ_t. If χ' and χ were incomparable, then neither $\chi' \leq \chi$ nor $\chi < \chi'$ and thus χ' violates *(1)* and *(2)*. If $\chi' < \chi$, then χ must have satisfied *(2)*, thus exist an upper bound σ_k incomparable to χ'; however, all upper bounds are comparable to χ' if it is a join *(1,2)*. Finally, if $\chi < \chi'$, then χ' must satisfy *(2)*, thus exist an upper bound σ_k incomparable to χ, failing analogously.