Electronic Communications of the EASST
Volume 41 (2011)

Proceedings of the Tenth International Workshop on Graph Transformation and Visual Modeling Techniques (GTVMT 2011)

A visual language for temporal specifications based on Spider diagrams

Paolo Bottoni, Andrew Fish

14 pages

Guest Editors: Fabio Gadducci, Leonardo Mariani
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

A visual language for temporal specifications based on Spider diagrams

Paolo Bottoni[1], **Andrew Fish**[2]

[1] Dipartimento di Informatica, "Sapienza" Università di Roma, Italy, [2]School of Computing, Engineering and Mathematics, University of Brighton, UK

Abstract:

Spider Diagrams are a well-established visual language to specify sets, their relationships, and constraints on their cardinalities. However, they do not support evolution of specifications, where one wants to state that under certain circumstances a specification becomes invalid and a new one must be used, nor transformation of specifications, where one needs operators to manipulate specifications. In this paper, we attack the first problem by developing a new system of timed Spider Diagrams which allow modellers to indicate the temporal range of validity of a specification. The approach is illustrated with examples of policies for library management.

Keywords: Visual constraint language, Time based systems, Spider diagrams.

1 Introduction

Spider Diagrams (SDs) are a diagrammatic logical specification and reasoning system built on top of Euler Diagrams (EDs), suitable for expressing monadic first order logic with equality statements [STHT04]. Constraint Diagrams [Ken97, FFH05] are an extension of SDs which are more expressive (introducing explicit syntax for the expression of quantification and relations), but come with the trade-off of more complexity in interpretation. They were proposed as a means for expressing constraints within software system modelling, potentially as a replacement for the textual Object Constraint Language (OCL) within the Unified Modeling Language (UML). These diagrams can be used to specify static constraints over the model such as a system invariant, or behavioural specifications in the form of pre and post-condition contracts. An advantage of using diagrammatic constraint languages is that users become able to display a logical proof as a sequence of diagrams, thereby enhancing confidence in its correctness.

However, these languages lack an explicit model of system evolution, as no notion of dynamics has been incorporated within them. In particular, time is not a primitive notion in SDs, so that time-dependent specifications are not directly definable. We develop here a diagrammatic system, called Timed Spider Diagrams that incorporates temporal constraints expressed as intervals over a time model based on *calendars* with reference to some granularity layering, e.g. hours, days, weeks. Intervals can span between some exact moments in calendar time, or define a duration from an onset. This facilitates the specifications of time-related policies, where some system state is required to last for some specific time and to be followed by some other state, again possibly with some specified duration. In particular, we develop the framework for a diagrammatic system, called Timed Spider diagrams that enables the visualisation of temporal

constraints by annotating Spider diagrams with temporal constraints on their validity, and discuss its properties. We provide a semantics for Timed SDs based on a natural representation of the evolution of the state of a system over an interval as a sequence of snapshots and use such a representation to check the consistency of policies specified through several diagrams, possibly presenting overlapping intervals. Besides developing the basis for such definitions, we use them to check if a diagram over an interval in a timed-SD-sequence is a valid instance of a model, or if it is a valid deduction from some such instance (exploiting standard inference figures for SD, as for example those in [HST05]). The proposed model can also be used to derive transformation rules from policies, to simulate the behaviour of processes complying with them, thus allowing both a static and a dynamic analysis of a specification. This opens new avenues of research, such as: (i) the expansion of diagrammatic logic results into a temporal setting; (ii) the translation of the underlying model of the temporal constraint language into a graph transformation setting.

In what follows, Section 2 introduces new notions of SD-specifications and instances used for modelling, their interpretations and the notion of an SD-instance satisfying a SD-specification. We introduce concepts related to intervals and interval specifications in Section 3, and introduce timed-SDs in Section 4, providing a notion of SD-stories and semantics of timed-SDs in terms of SD-stories. Finally, Section 5 discusses related work, whilst Section 6 draws conclusions.

2 Specialisation of SDs for modelling purposes

We adopt a variant to a standard definition of SDs and of their interpretation (see [HMT+01, HST05]), providing a more direct relation of SDs to modeling in object-oriented terms.

A (concrete) ED is a collection of (labelled) simple closed curves in the plane, called contours, decomposing it into connected *minimal regions*. A *zone* is a region inside one set of contours and outside all the other ones; zones may be *shaded*. A (concrete) SD is an ED together with extra syntax for *spiders*. These are trees whose vertices (called *feet*) are placed in zones, with no two vertices of the same tree lying within the same zone. All diagrams have a "boundary contour", drawn as a rectangle and labelled by U (the universe of discourse); all regions are then inside U.

A concrete diagram represents logical statements according to the following intuitive semantics. The interior of a contour represents a set (corresponding to the label) and the regions formed by taking intersection, union and complement of regions represent the result of the corresponding set operations. Spiders represent elements in the sets determined by their habitat (the smallest region containing the spider). Distinct spiders represent distinct elements. *Shading* places upper bounds on the cardinality of sets: there are no more elements in the set represented by a shaded zone than the number of spider feet touching that zone; a shaded zone without feet touching it represents an empty set. A single diagram is called *unitary*. Unitary systems can be extended to *compound* systems, allowing standard logical connectives between diagrams.

Figure 1 shows an example of two unitary Spider diagrams used for static constraint specification; taking the two diagrams in conjunction would yield a compound diagram. Together they state that the set of users of a library is formed of individuals of two different types: *reader* and *admin*. A reader can be in a state of *Active*, *Suspended* or *Banned* and active readers may be in only two states: either they have some loan in place, or they are considered idle. A different type of user, the administrator, is always active, without ever being in a state *WithLoan* or *IsIdle*.

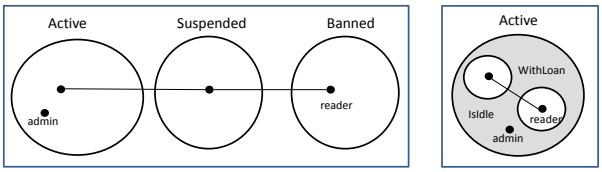

Figure 1: An example of two SD-specifications, stating two policy rules for a library.

Definition 1 Let $\mathscr{C}$ and $\mathscr{S}$ be disjoint sets of labels. A *unitary spider diagram* d on $\mathscr{L} = \mathscr{C} \cup \mathscr{S}$ is a tuple (C, Z, sh, S, h) such that: C is a set of curve labels drawn from $\mathscr{C}$, with $U \in C$; Z is a set of zones, each of which is defined as $z = (X, Y)$, where the sets X and Y form a partition of C (with $U \in X$ and Y possibly empty). X is referred to as the *inside* set of curves, and Y is the *outside* set; $sh : Z \to \mathbb{B}$ is a *shading* function assigning a boolean value to each zone. A zone z for which $sh(z) = true$ is said to be *shaded* and $Z^* \subset Z$ denotes the set of shaded zones; S is a set of spider labels drawn from $\mathscr{S}$; h is a *habitat* function $h : S \to \mathscr{P}(Z)$ that records the set of zones that each spider touches. Each unique pair $(s, z) \in S \times Z$ s.t. $z \in h(s)$ is called a *foot* of spider s, and we call F the set of all feet. A *compound spider diagram*, D, is any unitary SD or any construct with unitary SDs as primitives and allowing the unary logical negation operator *not*, and the binary logical conjunction and disjunction operators $\wedge$ and $\vee$ as connectives.

We provide a standard notion of interpretations and models (in a logic sense); see [HST05].

Definition 2 Let d be a SD on $\mathscr{L}$. An *interpretation* of d is a pair (U, ψ), where U is a set and ψ is a function that assigns subsets of U to each curve. The map ψ naturally extends to zones, and is also extended to spiders by mapping them to singleton subsets[1] of U. An interpretation is a *model* for d if it satisfies the *semantics predicate*, which states that: missing zones represent the empty set; each spider represents a unique element in the set represented by its habitat; shaded zones have cardinality at most the number of spiders touching that zone.

We distinguish between SDs used for system (invariant) specification and SDs used as model instances. However, we use the same syntax, with restrictions on the form of the labels. We situate SDs within object oriented specifications, with reference to a type system, assuming access to types, classes, and instances via labels. We interpret SD-specifications and SD-instances by specialising Definition 2, varying the interpretation of ψ on spiders in the two cases: for SD-specifications a spider represents a set of objects of a given type, whilst in an SD-instance a spider represents the singleton set containing the named object of a given type. Thus, SD-specifications place global constraints on a model (in a Software Engineering sense) instance.

Definition 3 An *SD-specification* is an SD, where $\mathscr{C}$ is drawn from the set of class or state names of the system and $\mathscr{S}$ from the set of types. An *SD-instance* is an SD, where $\mathscr{C}$ is drawn from the set of class or state names of the system, and $\mathscr{S}$ are pairs of object names and types. With d an SD-specification and d' an SD-instance, an *interpretation* of d or d' is a pair (U, ψ)

[1] For ease of exposition, we view singleton sets of U and the individual elements as interchangeable.

where U is restricted to be a set of objects of the system, and ψ maps curves to the set of objects of U of a certain class (as determined by the curve label). For an interpretation of d, ψ maps each spider to the set of objects of type indicated by its label. For an interpretation of d', ψ maps each spider to the singleton set containing the object with name and type specified by its label.

2.1 Satisfaction

For an SD-instance to satisfy an SD-specification, the spiders in the instance must only inhabit zones corresponding to those inhabited by spiders of that type in the specification, and no constraint on zone cardinality can be violated. Parts of the specification may be not relevant to the check, e.g. curves not present in the instance. Hence, a projection of a diagram d onto a set C, of abstract curves produces a diagram d^C with all curves not in C removed. This operation, besides redefining zones and their shading, modifies accordingly the habitat of spiders, coherently with the adopted semantics for these pieces of syntax. In Figure 2 (top) an SD-specification d combines the two policies in Figure 1. On the bottom left, an SD-instance d' indicates that the instance *John* of type `reader` is suspended. The bottom right shows the projection $d^{\{U,Suspended\}}$ of d onto the curve set of d'. As d' has a correctly typed individual (*John* and *Susan*) in an admissible zone for each type in $d^{\{U,Suspended\}}$, we say that d' *satisfies* d.

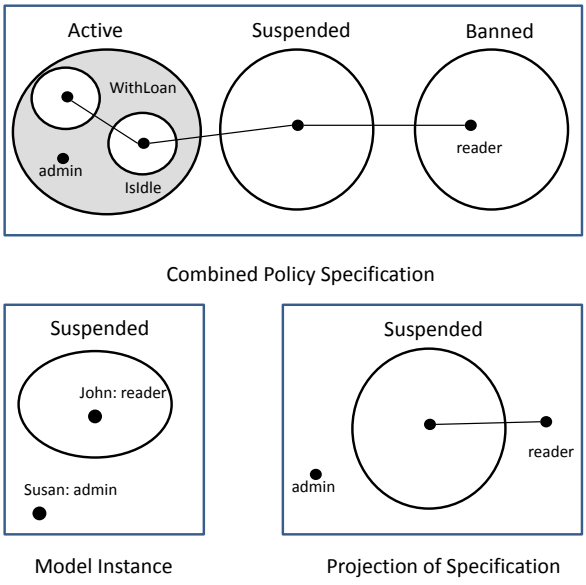

Figure 2: An example of an SD-instance.

Definition 4 Let $C \subseteq \mathscr{C}$ be a set of curves, with $U \in C$, and let $z = (X,Y) \in Z$ be a zone. The *projection* of z with respect to C, z^C, is the pair $(X \cap C, Y \cap C)$. This extends to the projection of Z with respect to C, denoted by Z^C. Let d be a SD with curve set C', where $C \subseteq C'$. If s is a spider in d with habitat $h(s)$, then the *projection* of s with respect to C is spider s^C with habitat $h(s^C) \subseteq Z^C$ such that $z \in h(s) \implies z^C \in h(s^C)$ and $z' \in h(s^C) \implies \exists z \in h(s)$ such that $z' = z^C$.

The projection of d with respect to C is the diagram, denoted d^C, obtained from d as follows: (1) we replace each zone of d, and each spider of d, by their projection with respect to C; (2) if $z \in Z(d^C)$ then z is shaded if and only if every zone $z' \in Z(d)$ such that $(z')^C = z$ is shaded.

When considering an SD-instance satisfying an SD-specification, we allow multiple spiders of the same type in the SD-instance, but insist that they all satisfy the constraints from the SD-specification. We also insist that in the SD-instance there is at least one spider of each type present in the SD-specification; moreover, given a spider in the SD-instance, its habitat is contained in the projection of the habitat of the spider for that type in the SD-specification.

Definition 5 Let d_1 be an SD-specification and let d_2 be an SD-instance[2]. We say that d_2 *satisfies* d_1, denoted $d_2 \models d_1$, if: 1) $C_2 \subseteq C_1$; 2) $Z_2 \subseteq (Z_1)^{C_2}$; 3) $Z_2^* \subseteq (Z_1^*)^{C_2}$; 4) $(Z_1^*)^{C_2} \cap (Z_2 \setminus Z_2^*) = \emptyset$; 5) $\pi_2(S_2) = S_1$, where π_i takes the i-th component of a tuple and is extended to sets; 6) $\forall s_2 \in S_2[\exists s_1 \in S_1[\pi_2(s_2) = s_1 \wedge h_2(s_2) \cap Z_1^{C_2} \subseteq h(s_1)^{C_2}]]$.

3 Interval specifications

We extend SDs to enable temporal specifications, associating temporal annotations with the elements of a SD or with a whole SD, to indicate the time over which the associated constraint/situation holds. While several models of time could be utilised, we choose to use a model based on calendar time, where consecutive integer indexes are used to indicate contiguous time-intervals each with a duration defined by a *granule* or time-unit. The intervals we associate with diagrams have integer endpoints and are interpreted as the sequence of consecutive granules indexed by the integers within the interval, including the endpoints. We are interested in the *meet*, *during*, and *overlap* relations of Allen's interval algebra [AF94], and they are adapted for use with the intervals associated with the diagrams. One can do this by considering Allen's relations for the actual time-intervals that are the union of the time-intervals of the consecutive integers in the interval associated to the diagram. The simple set-up presented here essentially combines the model of [AF94] with the calendar model adopted in [BBF01].

We use $\mathbb{N}$ to denote the set of natural numbers and $\mathbb{N}_0$ for $\mathbb{N} \cup \{0\}$. For the granularity of intervals we consider standard time units e.g. seconds, hours, days, with the usual layering among them. Hence, each granule can be decomposed into finer sub-granules up to some undecomposable granule (i.e. our time model is ultimately a discrete one). However, we consider that significant specifications are expressed with respect to decomposable granules. Time 0 refers to the system starting time for the model at hand.

Definition 6 Given a time unit u, let $a, b \in \mathbb{N}_0$ and $a \leq b$. We call $[a,b]_u$ the *time interval in* u, i.e. the ordered sequence of granules indexed by all natural numbers between and including a and b. The set of all time intervals in u is denoted by $\mathscr{I}_u$. With $I_1 = [a_1,b_1]_u, I_2 = [a_2,b_2]_u \in \mathscr{I}_u$, we say: 1) I_1 *meets* I_2, denoted by $I_1 \sqcap I_2$, iff $a_2 = b_1 + 1$; 2) I_1 *during* I_2, denoted by $I_1 \sqsubseteq I_2$ iff

[2] For simplicity we assume here that these are unitary diagrams and that the SD-instance does not contain curves that are not present in the SD-specification.

$a_1 \geq a_2$, $b_1 \leq b_2$[3]; 3) I_1 *overlaps* I_2, denoted by $I_1 \sqcup I_2$, iff $a_1 < a_2 \leq b_1 < b_2$. If none of the above occurs, we say that I_1 and I_2 are *disjoint*, with the two cases $I_1 < I_2$ and $I_1 > I_2$. For non disjoint I_1 and I_2, we define their *merge* $I_1 \odot I_2$ as the interval $I = [min(a_1,a_2), max(b_1,b_2)]$.

Example 1 *We have* $[0,2]_d \sqcap [3,6]_d$*; they merge to give* $[0,6]_d$*. In this case we use days as granules, so the interval specifies the 7 consecutive days of a week; the interval* $[1,1]_d$*, with the same start and end point, is interpreted as the second day of the week, starting counting at 0. We also have* $[1,2]_d \sqsubseteq [1,4]_d$*, and* $[1,2]_d \odot [1,4]_d = [1,4]_d$*;* $[1,3]_d \sqcup [2,4]_d$ *and* $[1,3]_d \odot [2,4]_d = [1,4]_d$*;* $[1,2]_d$ *is disjoint from* $[4,5]_d$*.*

Besides simple intervals, we have interval specifications involving expressions on variables.

Definition 7 Given a time unit u, an *interval specification in* u is a construct $[exp_1, exp_1 + exp_2]_u$, where exp_1 and exp_2 are two linear expressions including natural numbers and variables (with integer coefficients other than zero) that can be evaluated over $\mathbb{N}_0$. $Var(exp)$ denotes the set of variables appearing in exp and $Var(I)$ the cumulative set of variables from exp_1 and exp_2. A valuation $Val(exp)$ is a set of assignments to natural numbers for each variable in $Var(exp)$, so that each occurrence of the same variable name is assigned the same value. Analogously $Val(I)$ denotes the simultaneous valuation of exp_1 and exp_2. The value of exp according to $Val(exp)$ is denoted $\|Val(exp)\|$. The *interpretation* of an interval specification $tSpec = [exp_1, exp_1 + exp_2]_u$ is the set of intervals $int(tSpec) = \{[a,b]_u \in \mathscr{I}_u \mid \exists Val(tSpec), s.t. \|Val(exp_1)\| = a, \|Val(exp_2)\| = b - a\}$. We denote by $\mathscr{I}'_u$ the set of interval specifications in u.

Example 2 *Consider an interval specification* $I = [x+1, x+1+2y]_u$*. Then* $Var(x+1) = \{x\}$*,* $Var(2y) = \{y\}$*, and* $Var(x+1+2y) = \{x,y\} = Var(I)$ *are sets of variables. Suppose that we have valuation functions* $Val(x+1)$ *such that* $x \mapsto 3$*, and* $Val(2y)$ *such that* $y \mapsto 1$*. Then* $x \mapsto 3, y \mapsto 1$ *is an assignment of natural numbers to variables in* $Var(I)$*. The values of the expressions according to these assignments are:* $||Val(x+1)|| = 4$*,* $||Val(2y)|| = 2$*,* $||Val(x+1+2y)|| = 6$*,* $||Val(I)|| = [4,6]_u$*. The interpretation of* $[x+1, x+1+2y]_u$ *is the set of all intervals* $[a,b]_u$ *s.t.* $a \geq 1$ *and* $b-a$ *is an even number (possibly 0). Note that each valuation of an interval specification,* I*, fixes it to be a specific interval.*

In the following, we omit the indication of the unit where no ambiguity arises, and deal only with specifications $tSpec$ s.t. $int(tSpec) \neq \emptyset$, ruling out specifications such as $[x, x-(2x+1)]$. Two non disjoint intervals I_1, I_2 in a set of intervals I can be naturally decomposed into a sequence of at most three contiguous intervals (i.e. meeting and not overlapping) as follows: 1) if $I_1 \sqcap I_2$ do nothing; 2) if $I_1 \sqsubseteq I_2$ (and it is not the case that I_1 starts, finishes or is equal to I_2, hence $a_1 > a_2$ and $b_1 < b_2$) then build the sequence $[a_2, a_1 - 1], [a_1, b_1], [b_1 + 1, b_2]$ (the other cases being easily derivable); 3) if $I_1 \sqcup I_2$, then build the sequence $[a_1, a_2 - 1], [a_2, b_1], [b_1 + 1, b_2]$. This procedure can be iterated on any set of intervals (s.t. no interval is disjoint from all the others) until each non disjoint pair of intervals meet. It can be proved that the resulting collection of intervals is unique. Note that this does not create new end or start points other than those in the set $\{a_2 - 1, \ldots, a_n - 1, b_1 + 1, \ldots, b_{n-1} + 1\}$. Also note that some intervals can be reduced to a

[3] For our purposes, we include in this the special cases defining the relations *finishes*, *starts*, or *equal* in [AF94].

unitary interval, e.g if $a_2 - 1 = a_1$ for case 2) above. We extend the decomposition concept to interval specifications.

Definition 8 Let $I = \{I_1, \ldots, I_n\}$ be a finite set of intervals, for $I_i = [a_i, b_i] \in \mathscr{I}$, s.t. $\bigodot_{i\in\{1,\ldots\ n\}} I_i$ is defined and equal to $[min(a_i), max(b_i)]$ for $i \in \{1, \ldots, n\}$. A *non overlapping cover* of I is a finite set of intervals $J = \{J_1, \ldots, J_m\}$, with $J_k = [c_k, d_k]$, s.t.: 1) $\bigodot_{i\in\{1,\ldots,n\}} I_i = \bigodot_{j\in\{1,\ldots,m\}} J_j$; 2) for each $k \in \{1, \ldots, m-1\}$, J_k meets J_{k+1} (i.e. $c_{k+1} = d_k + 1$). J is the *canonical non overlapping cover* if its set of intervals coincides with the one derived from the procedure described above. Let $I' = \{I'_1, \ldots, I'_n\}$ be a finite set of interval specifications, and let Val be a valuation function for I' (i.e. for all of the interval specifications in I'). Then a *(canonical) non overlapping cover* of I' w.r.t. Val is a finite set of interval specifications $J' = \{J'_1, \ldots, J'_m\}$ s.t. $\|Val(J')\|$ is a (canonical) non overlapping cover of $\|Val(I')\|$.

Example 3 *Suppose that* $I = \{I_1 = [1,5], I_2 = [4,5], I_3 = [2,9]\}$. *Then* $\bigodot_{i\in\{1,2,3\}} I_i = [1,9]$. *Let* $J = \{J_1 = [1,1], J_2 = [2,3], J_3 = [4,5], J_4 = [6,9]\}$, *so* $\bigodot_{k\in\{1,2,3,4\}} J_k = [1,9]$ *and* J *is the canonical non-overlapping cover of* I. *On the contrary,* $J' = \{[1,2],[3,4],[5,6],[7,9]\}$ *is another non-overlapping cover, which is not canonical as can be easily seen since 7 is an endpoint of* J' *not in the set* $\{1,2,3,4,5,6,9\}$.

4 Timed Spider Diagrams

We introduce timed SDs as associations of SDs with specifications of admissible time intervals for elements in a diagram and with constraints on variables appearing in the specifications. They can express fairly complex temporal relations and we wish to facilitate operations such as the combination of compatible timed-SDs (e.g. stating two parts of a same policy).

Figure 3 shows a compound diagram and introduces variables in SD-specifications. Variables are used to specify intervals which can start at any time, so that a designated variable is instantiated to define the onset of the interval. Variables are bound to natural numbers, and subject to constraints. The figure refines the library policy: a user who stays in the *Suspended* state for a whole period of 30 days (for example for not paying a fine) becomes banned. In this case the designated variable x can be instantiated at any time, in correspondence with the moment where a user enters the *Suspended* state, while the use of k and of the associated constraint indicates that being in this state may cease at any time before the deadline of 30 days[4]. We consider time-valid diagrams where elements are present only if elements on which they depend are also present, e.g. a foot can be in a zone only during the existence interval for that zone.

Definition 9 A *timed SD* is a construct $d_T = (d, \omega, X)$, where $d = (C, Z, sh, S, h)$ is an SD, $\omega : C \cup Z \cup (Z \times \mathbb{B}) \cup S \cup F \to \mathscr{I}'$ is a function assigning an interval specification to each object[5] in d, and X is a set of linear constraints on the valuations for the specifications assigned by ω, where all instances of variables with the same name receive the same value. Let $Val(\omega)$ be a valuation function that consistently evaluates every variable in the image of ω. A valuation

[4] For simplicity, we omit here the indication of the time unit.
[5] F is the set of feet as derived from $h : S \to \mathscr{P}(Z)$.

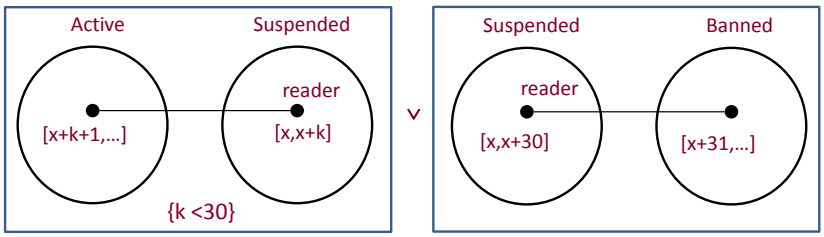

Figure 3: A user either exits from suspension within 30 days or becomes banned.

$\|Val(\omega(x))\| = [a,b]$ satisfies X, denoted $\|Val(\omega(x))\| \models X$, if no assignment in Val violates a constraint in X, and $a \leq b$. We say that d_T is *time-valid* if $\exists Val(\omega)$ s.t.:

1. $\forall x \in C \cup Z \cup (Z \times \mathbb{B}) \cup S \cup F[\|Val(\omega(x))\| \models X]$.

2. The following general constraints hold:

   - $\forall f = (s,z) \in F[\|Val(\omega(f))\| \sqsubseteq \|Val(\omega(s))\|]$;
   - $\forall f = (s,z) \in F[\|Val(\omega(f))\| \sqsubseteq \|Val(\omega(z))\|]$;
   - $\forall z \in Z[\|Val(\omega(z,sh(z)))\| \sqsubseteq \|Val(\omega(z))\|]$;
   - $\forall z \in Z, c \in C[\|Val(\omega(z))\| \sqsubseteq \|Val(\omega(c))\|]$.

Given an interval specification $[exp_1, exp_1 + exp_2]$, if there exists a variable $x \in Var(exp_2) \setminus Var(exp_1)$, s.t. no constraint is explicitly given for x, then we say that $exp_1 + exp_2$ is *unlimited* and write $[exp_1, \ldots]$. In the following, we only deal with time-valid diagrams, and rule out inadmissible sets of specifications, such as $I = \{[x, x+y], [z-y, x-z-1]\}$ with $X = \{y = z, z = 2x\}$.

Elements in a diagram are present during the specified time-intervals, with any non-assigned element being assigned $[0, \ldots]$ by default. We interpret that for any interval I disjoint from the image of a diagram element, that element is not present during I. Note that associating an interval with a pair in $Z \times \mathbb{B}$ allows for modifications over time of the property of being shaded for a zone.

A timed-SD (d, ω) can represent a complex time-based set of constraints. However, this can be reduced to a collection of *basic timed SDs* where each diagram is associated with a single interval. This construction is possible if ω is well-defined for every element of d and the image of any such element under ω is a single interval. Hence, for each time-valid timed SD there is an equivalent sequence of basic timed SDs, where the intervals of consecutive diagrams are contiguous. In Definition 10, we first define these concepts for SDs, permitting specialisation to SD-specifications and SD-instances, and then we make use of specialisations by providing semantics to the timed SD-specifications.

Definition 10 (1) A *basic timed-SD* is a tuple (d, I, X), where d is an SD, $I \in \mathscr{I}'$ and X is a set of linear constraints on $Var(I)$. (2) Let $\langle d \rangle = \langle (d_1, I_1, X_1), \ldots, (d_n, I_n, X_n) \rangle$ be a sequence of basic timed-SDs. We say that $\langle d \rangle$ is *contiguous* w.r.t. a joint valuation function Val over all interval specifications $I_1, \ldots, I_n$, if $||Val(I_j)|| \sqcap ||Val(I_{j+1})||$ for all $j \in \{1, \ldots, n-1\}$.

A timed SD encapsulates complex temporal constraints. For purposes such as to allow modularisation of the constraints according to some time-line, as well as defining the semantics of timed SD-specifications, we provide a conversion from timed-SDs to sequences of contiguous basic timed-SDs. The following can be applied to SD-specifications or SD-instances.

Definition 11 Let $d_T = (d, \omega, X)$ be a time-valid timed SD w.r.t. a valuation function $Val(\omega)$. Let $\langle d' \rangle = \langle (d'_1, J_1, Y_1) \ , \ldots, \ (d'_m, J_m, Y_m) \rangle$ be a sequence of basic timed SDs, and Val' a joint valuation function over the J_i's s.t.: (i) $\langle d' \rangle$ is contiguous w.r.t. Val'; (ii) for each subinterval I of $\|Val'(J_i)\|$, d'_i consists of exactly the diagram elements x of d for which $I \sqsubseteq \|Val(\omega(x))\|$. Then d' is a *time-decomposition* of d_T. If m is minimal subject to (i) and (ii), then d' is a *canonical time-decomposition* of d_T.

Algorithm 1 Let $d_T = (d, \omega, X)$ be a time-valid timed SD, and let Val be a valuation function over ω. Then: 1) Let I be the set of all of the interval specifications obtained as $\omega(x)$, for any $x \in C \cup Z \cup (Z \times \mathbb{B}) \cup S \cup F$. 2) Construct $J = \{J_1, \ldots, J_m\}$, the canonical non overlapping cover of I w.r.t Val. 3) Construct $\langle (d_1, J_1, Y_1), \ldots, (d_m, J_m, Y_1) \rangle$, where d_i is the SD consisting of the set of diagram elements x, for all $x \in C \cup Z \cup (Z \times \mathbb{B}) \cup S \cup F$ for which $\|Val(J_i)\|$ is not disjoint from $\|Val(\omega(x))\|$.

Theorem 1 *Let $d_T = (d, \omega, X)$ be a time-valid timed-SD, and let Val be a valuation function over ω. Then the construct $\langle (d_1, J_1, Y_1), \ldots, (d_m, J_m, Y_1) \rangle$ from Algorithm 1 is a* canonical time-decomposition *of d_T.*

Proof. Post-valuation, one can consider the set of intervals associated to each diagram element, and decompose this into a sequence of contiguous intervals which collectively merge to yield the whole timeline. Then, for each interval in this decomposed timeline, there is a single corresponding diagram (comprising all and only the diagram elements of d that are present within that interval); each of these is a well-defined diagram since d_T is time-valid. These diagrams together with associated intervals constitute a contiguous sequence of basic timed-SDs. It follows that they form a canonical time-decomposition of d_T, due to the construction. The argument lifts from intervals to interval specifications. □

4.1 Satisfaction and semantics of timed-SDs

We extend the notion of satisfaction of an SD-specification by an SD-instance to timed SDs. That is, we define what it means for a timed SD-instance to satisfy a timed SD-specification. We do this by first translating both specification and instance into distinct contiguous sequences of basic timed SDs. Then we provide a definition of satisfaction of a sequence of contiguous basic timed SD-specifications by a contiguous sequence of basic timed SD-instances. This is done by checking that over every interval of the sequence of basic SD-instances, the SD-instance satisfies the SD-specification during that interval. A sequence of contiguous basic timed SD-instances satisfying a contiguous sequence of (basic) timed SD-specifications is called a story and the semantics of a timed SD-specification is defined to be the set of all of its stories. This could also be viewed as the set of all possible snapshot sequences that satisfy all of the constraints for the corresponding intervals.

Definition 12 Let $\langle d \rangle = \langle (d_1, I_1, X_1), \ldots, (d_n, I_n, X_n) \rangle$ be a contiguous sequence of basic timed SD-specifications, and let $\langle d' \rangle = \langle (d'_1, J_1, Y_1), \ldots, (d'_m, J_m, Y_m) \rangle$ be a contiguous sequence of basic timed SD-instances. Then, for a common valuation function Val, over $I_1, \ldots, I_n, J_1, \ldots, J_m$ and an interval K, we say that: (i) $\langle d' \rangle \models_K \langle d \rangle$, if for i, j s.t. $K, \|Val(I_i)\|$, and $\cap \left\|Val(J_j)\right\|$ are jointly overlapping, $d'_j \models d_i$; (ii) $\langle d' \rangle$ *satisfies* d, denoted $\langle d' \rangle \models \langle d \rangle$, if $\forall J_i \in \{J_1, \ldots, J_m\}$, we have $\langle d' \rangle \models_{J_i} \langle d \rangle$. Let $d_T = (d, \omega, X)$ be a time-valid timed-SD-specification (w.r.t. Val) and let $\langle d'' \rangle = \langle (d''_1, K_1, Z_1) \ , \ldots, \ (d''_p, K_p, Z_p) \rangle$ be a time-decomposition of d_T (w.r.t. Val''). Then, for a common valuation function Val' over $K_1, \ldots, K_p, J_1, \ldots, J_m$, $\langle d' \rangle$ *satisfies* d_T if $\langle d' \rangle$ satisfies $\langle d'' \rangle$.

Definition 13 Let $d_T = (d, \omega, X)$ be a time-valid timed-SD-specification. Let $\langle d \rangle = \langle (d_1, I_1, X_1), \ldots, (d_n, I_n, X_n) \rangle$ be a contiguous sequence of basic timed SD-instances, s.t. for each valuation function Val of ω, there is an extension of Val to Val_2, a valuation of $\omega, I_1, \ldots, I_n$ with $\langle d \rangle \models d_T$, w.r.t. Val_2. Then $\langle d \rangle$ is called a d_T-story and the *semantics* of d_T is the set of all d_T-stories. We also speak of SD-stories when d_T is implicit.

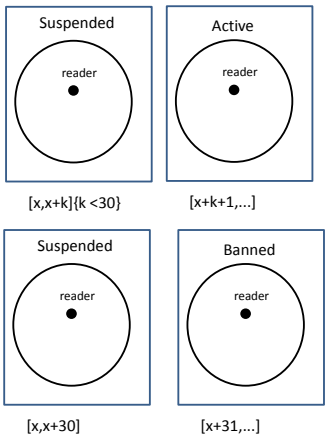

Figure 4: (a) Alternative cases which are contiguous basic timed SD-specifications for the timed SD-specification of Figure 3.

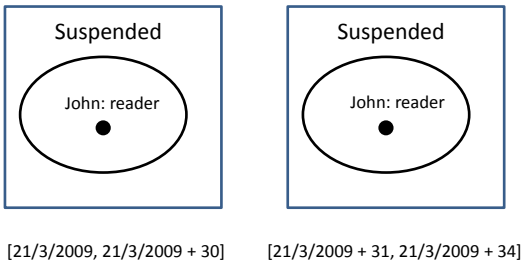

Figure 4: (b) A contiguous sequence of basic timed SD-instances which is not an SD-story for the specification of Figure 3.

Figure 4(a) shows examples of two contiguous sequences of basic timed SD-specifications that encapsulate the alternative cases of the timed SD-specification of Figure 3. Figure 4(b) shows an example of contiguous sequence of basic timed SD-instances which does not satisfy either of the sequences in Figure 4(a), and is not an SD-story for Figure 4(a); note that if the *Suspended* state in the second diagram were changed to a *Banned* state, then this would by a story for the SD-specification in Figure 4(a).

Any SD-instance satisfying a canonical time decomposition of a timed SD-specification is an SD-story. There could be many of these SD-instances, but there is at least one. The canonical time decomposition of a timed-SD exists and is unique, provided we place certain restrictions on the valuation function.

Theorem 2 *Let $d_T = (d, \omega, X)$ be a time-valid timed SD-specification, w.r.t. Val, a valuation function over ω. Let $\langle d' \rangle = \langle (d'_1, I_1, X_1), \ldots, (d'_n, I_n, X_n) \rangle$ be a contiguous sequence of basic timed SD-instances that satisfies a canonical time decomposition of d_T. Then $\langle d' \rangle$ is an SD-story (called a* canonical SD-story*).*

Proof. Satisfaction of a canonical time decomposition of d_T implies satisfaction of d_T. □

Corollary 1 *Each time-valid timed SD-specification $d_T = (d, \omega, X)$ has a non-empty semantics.*

Proof. Since d_T is time-valid, there is a canonical time decomposition of d_T. Any SD-instance which realises the spider types of this canonical time decomposition as appropriate (name,type) pairs is an SD-story, as required[6]. □

Definition 14 An *initialised canonical time decomposition* of d_T is a canonical time decomposition of d_T, w.r.t. a valuation function *IVal*, which assigns the minimal admissible value to $iexp_1$, with $J_1 = [iexp_1, iexp_1 + iexp_2]$. An *initialised left-minimal canonical time decomposition* is an initialised canonical time decomposition with valuation *ILMVal* such that the lengths of each J_i is minimal over all *IVal* valuations, miminizing in order of increasing index i.

Theorem 3 *Let $d_T = (d, \omega, X)$ be a time-valid timed SD, and let Val be a valuation function over ω. Then there exists a unique initialised left-minimal canonical time decomposition.*

Proof. Existence derives from Theorem 2, uniqueness from the minimization process. □

Theorem 4 *Let d_T be a timed-SD specification. Then the time-validity of d_T is decidable.*

Proof. The interval specification constraints reduce to a system of linear diophantine equations (relating end-points to start-points) under a set of constraints. That the solution, without constraints, is decidable is a classical result. □

5 Related work

Several models of time have been proposed for formal specifications, both in relation to real-time [GB03, AD94] or hybrid [Hen96] behaviours. Time-based extensions have been also proposed for calculi or specification languages of concurrent processes (see [FOP09] or [BW03]). In general, these models deal with intervals to model uncertainty about the actual occurrence of an event. In Statemate, also a clock-synchronous semantics is provided where events can only occur when a clock ticks [EJW02]. This view was adopted also in [GVH03] to integrate time in graph transformations, by introducing a specific attribute updated by clock messages to processes.

In UML, a simple model of time is adopted, based on a notion of observation, and able to express durations and deadlines [OMG10], whereas in the profile for real-time applications, effects connected with latencies in observation can be taken into account [OMG05]. In both cases, OCL constraints can incorporate conditions on time expressions.

[6] We assume the set U has sufficient elements for such a realisation.

In general, we are interested here in the persistence in a state over a period as dictated by time-dependent policies, rather than in modeling the occurrence of specific transitions triggered by any type of events. As a consequence, the model of time adopted here is connected to the notion of calendar time, as adopted in the area of temporal databases and temporal rule based access control. In particular, we adopt a model analogous to that Bertino *et al.* [BBF01], based on a formalism proposed by Niezette and Stevenne [NS92], which however considered intervals of fixed length, repeated after some time. In Bertino's model a calendar is a set of contiguous intervals, each with its own duration, containing all the instants between its extreme granules, from the start of the first granule to the end of the second one. Based on this, they introduce periods to express that some roles have to be granted specific access rights at recurring times. Ning *et al.* exploit the notions of calendars and granules to define a calendar algebra, where operations allow the grouping of intervals or the subdivision of granules [NWJ02]. Of interest here is the notion that the intervals covered by distinct granules (at the same level) cannot be interleaved. Others consider calendar times as corresponding to an instant, rather than a granule [KÖ95]. A vast examination of the problems related to the use of different granularities is in [EM05].

The field of multimedia is another area in which the modeling of time is relevant, in particular as sequential media (typically audio and video) may have to be synchronised with the presence of static documents for some time. In many cases one is therefore interested in considering durations of intervals which can start at any point in time, rather than at specific instants. As an example, Bowman *et al.* have defined a formalism in which, once a starting point for the system is set, reasoning can be performed on the occurrence, within the current interval, of a state, based on the lengths of the current and previous intervals [BCKT03].

In addition to considering intervals, in the approach presented in the paper, the use of instants in the specification of rules can provide a weak form of clock-synchronous specification, associated with the triggering of a time-dependent transition.

In the field of SDs, to our knowledge this is the first attempt to integrate time-related aspects in the formalism. Moreover, we draw a more precise correspondence with object-oriented modeling, by distinguishing specifications from instance models and providing two distinct interpretations for spiders, as types and as typed individuals. A relation can be drawn with the construction of parallel models in Z for constraint diagrams in [HS05].

The idea of representing system dynamics through sequences of EDs was introduced in [BF10], to follow the evolution of sets (rather than the state of individuals) under the effect of a Reaction Systems [ER07], and colour was used to assist in tracking families of sets through the sequences.

6 Discussion and conclusions

We have proposed an extension of SDs which enables them to express time-dependent policies, in which curves indicate the permanence of individuals, modeled by spiders, in some state, over some interval. Stories of individual evolution can then be checked against these specifications to assess conformance to the policy, so that the semantics of a policy specification is the set of stories conformant to that specification. We adopt a simple model of time, related to measurement units rather than system clocks. The model is suitable to the expression of requirements and constraints on system configurations, rather than of real-time behaviours.

However, a limited form of dynamics can be provided by the use of rules enforcing modifications in the state of individuals according to a policy. In this case, we could specify rules over timed specifications, to produce dynamic views of a specified system. To this end, we could use rules triggered by the onset or offset of an interval (in this case making reference to the mapping of intervals onto the fundamental granule layer). For example, given a collection of specifications $\langle (d_1, \omega_1), \ldots, (d_n, \omega_n) \rangle$ annotated with contiguous interval specifications, one can derive a collection of rules as pairs $((d_i, \omega_i), d_{i+1})$, together with some mapping μ from elements of d_i to elements of d_{i+1}. The interpretation of such a rule would be that if a system has been described by an SD instance satisfying d_i over an interval given by a valuation of ω_i, then at the end of this interval it moves to a state described by an SD instance, related to the previous one via μ, which is a model for d_{i+1}.

We plan to extend this work in a number of directions. Firstly, standard notions and results from the theory of (static) SDs have to be reviewed and lifted to timed SDs, taking into account the distinction between spiders at the different levels. Secondly, the extension to real-time may require a different basis for time, considering open intervals over the reals. Thirdly, we plan to consider different types of dynamics, integrating event-based and time-dependent specifications, exploiting the mapping of SDs to typed attributed graphs, called *Spider Graphs*, presented in [BFP10], possibly following the approach in [GVH03].

Bibliography

[AD94] R. Alur, D. L. Dill. A Theory of Timed Automata. *TCS* 126(2):183–235, 1994.

[AF94] J. F. Allen, G. Ferguson. Actions and Events in Interval Temporal Logic. *J. Log. Comput.* 4(5):531–579, 1994.

[BBF01] E. Bertino, P. A. Bonatti, E. Ferrari. TRBAC: A temporal role-based access control model. *ACM Trans. Inf. Syst. Secur.* 4(3):191–233, 2001.

[BCKT03] H. Bowman, H. Cameron, P. King, S. Thompson. Mexitl: Multimedia in Executable Interval Temporal Logic. *Formal Methods in System Design* 22:5–38, January 2003.

[BF10] P. Bottoni, A. Fish. Coloured Euler diagrams: a tool for visualizing dynamic systems and structured information. In *Proc. Diagrams 2010*. LNAI 6170, pp. 39–53. 2010.

[BFP10] P. Bottoni, A. Fish, F. Parisi-Presicce. Preserving constraints in horizontal model transformations. *GTVMT-2010, ECEASST* 29:1–14, 2010.

[BW03] V. Bulitko, D. C. Wilkins. Qualitative simulation of temporal concurrent processes using Time Interval Petri Nets. *Artificial Intelligence* 144(1-2):95 – 124, 2003.

[EJW02] R. Eshuis, D. N. Jansen, R. Wieringa. Requirements-Level Semantics and Model Checking of Object-Oriented Statecharts. *Requir. Eng.* 7(4):243–263, 2002.

[EM05] J. Euzenat, A. Montanari. Chapter 3 Time granularity. In M. Fisher and Vila (eds.), *Handbook of Temporal Reasoning in Artificial Intelligence*. Foundations of Artificial Intelligence 1, pp. 59–118. Elsevier, 2005.

[ER07] A. Ehrenfeucht, G. Rozenberg. Events and modules in reaction systems. *TCS* 376:316, 2007.

[FFH05] A. Fish, J. Flower, J. Howse. The Semantics of Augmented Constraint Diagrams. *JVLC* 16:541–573, 2005.

[FOP09] M. Falaschi, C. Olarte, C. Palamidessi. A framework for abstract interpretation of timed concurrent constraint programs. In *Proc. PPDP '09*. Pp. 207–218. ACM, 2009.

[GB03] H. Giese, S. Burmester. Real-Time Statechart Semantics. Technical report tr-ri-03-239, University of Paderborn, 2003.

[GVH03] S. Gyapay, D. Varro, R. Heckel. Graph Transformation with Time. *Fundamenta Informaticae* 1:1–22, 2003.

[Hen96] T. A. Henzinger. The Theory of Hybrid Automata. In *LICS*. Pp. 278–292. 1996.

[HMT+01] J. Howse, F. Molina, J. Taylor, S. Kent, J. Gil. Spider Diagrams: A Diagrammatic Reasoning System. *JVLC* 12(3):299–324, 2001.

[HS05] J. Howse, S. Schuman. Precise Visual Modelling. *SoSyM* 4:310–325, 2005.

[HST05] J. Howse, G. Stapleton, J. Taylor. Spider Diagrams. *LMS Journal of Computation and Mathematics* 8:145–194, 2005.

[Ken97] S. Kent. Constraint Diagrams: Visualizing Invariants in Object Oriented Modelling. In *Proc. OOPSLA97*. Pp. 327–341. ACM Press, October 1997.

[KÖ95] A. Kurt, Z. M. Özsoyoglu. Modeling and Querying Periodic Temporal Databases. In *Proc. DEXA Workshop*. Pp. 124–133. 1995.

[NS92] M. Niezette, J. Stevenne. An efficient symbolic representation of periodic time. In *Proc. CIKM 1992*. Pp. 161–168. 1992.

[NWJ02] P. Ning, X. S. Wang, S. Jajodia. An Algebraic Representation of Calendars. *Annals of Mathematics and Artificial Intelligence* 36(1-2):5–38, 2002.

[OMG05] OMG. UML Profile for Schedulability, Performance, and Time Specification, Version 1.1. Technical report formal/05-02-06, OMG, 2005. http://www.omg.org/cgi-bin/doc?realtime/05-02-06.pdf.

[OMG10] OMG. OMG Unified Modeling Language (OMG UML), Superstructure Version 2.3. Technical report formal/2010-05-05, OMG, 2010. http://www.omg.org/spec/UML/2.3/Superstructure.

[STHT04] G. Stapleton, S. Thompson, J. Howse, J. Taylor. The Expressiveness of Spider Diagrams. *J. of Logic and Computation* 14(6):857–880, December 2004.