The Tale of SOLOIST: A Specification Language for Service Compositions Interactions

Domenico Bianculli[1], Carlo Ghezzi[2], and Pierluigi San Pietro[2]

[1] University of Luxembourg - SnT Centre, Luxembourg
domenico.bianculli@uni.lu
[2] Politecnico di Milano - DEI - DEEP-SE Group, Italy
{carlo.ghezzi,pierluigi.sanpietro}@polimi.it

Abstract. Service-based applications are a new class of software systems that provide the basis for enterprises to build their information systems by following the principles of service-oriented architectures. These software systems are often realized by orchestrating remote, third-party services, to provide added-values applications that are called service compositions. The distributed ownership and the evolving nature of the services involved in a service composition make verification activities crucial. On a par with verification is also the problem of formally specifying the interactions—with third-party services—of service compositions, with the related issue of balancing expressiveness and support for automated verification.

This paper showcases SOLOIST, a specification language for formalizing the interactions of service compositions. SOLOIST has been designed with the primary objective of expressing the most significant specification patterns found in the specifications of service-based applications. The language is based on a many-sorted first-order metric temporal logic, extended with new temporal modalities that support aggregate operators for events occurring in a certain time window. We also show how, under certain assumptions, the language can be reduced to linear temporal logic, paving the way for using SOLOIST with established verification techniques, both at design time and at run time.

1 Introduction

Modern-age software engineering has to deal with novel kinds of software systems, which exhibit new features that often demand for rethinking and extending the traditional methodologies and the accompanying methods and techniques. One class of new software systems is constituted by *open-world* software [5], characterized by a dynamic and decentralized nature; service-based applications (SBAs) represent an example of this class of systems. SBAs are often defined as service compositions, obtained by orchestrating—with languages such as BPEL [2]—existing services, possibly offered by third-parties. This kind of applications has seen a wide adoption in enterprises, which nowadays develop their information systems using the principles of service orientation [20].

C.S. Păsăreanu and G. Salaün (Eds.): FACS 2012, LNCS 7684, pp. 55–72, 2013.

The development of SBAs is usually spread across multiple organizations or multiple divisions within a single organization, and promotes a loose organizational coupling between service providers and service integrators. Moreover, in open-world software, changes are frequent, unexpected, and welcome [29]. On one hand, services are developed, deployed, operated, and evolved (e.g., by changing the interface, the implementation, the business protocol, or the quality of service guarantees) autonomously by service providers. On the other hand, service integrators may leverage dynamic binding as well as self-adaptation techniques to change a composite service at run time. These factors lead to a distributed ownership and to an evolving nature of service compositions, which affect the notions of correctness, dependability and in general all quality attributes of SBAs.

Guaranteeing quality attributes of SBAs poses new challenges to the definition of verification methodologies. This challenge has been taken on in the last years by the research community, which has proposed several techniques for the verification of SBAs, both at design time and at run time; see for example [4,10,35]. Equal in importance to verification techniques are the specification languages used to express the requirements of the service interactions that one wants to check. In most of the cases, the specification language is some logical language, such as the CTL and LTL temporal logics and the Event Calculus, or a domain-specific language defined to represent some non-functional attributes (e.g., response time).

Despite significant advances in research and in prototype implementation, these approaches did not spread to the world of practitioners. One of the reasons is that proposed specification languages do not meet the expressiveness requirements of SBAs. In a previous work [8], some of the authors performed an extensive analysis of requirements specifications of SBAs, written both in research settings and in industrial settings, to characterize the use of property specifications patterns in SBAs. The results of this study showed that: a) the majority of requirements specifications stated in industrial settings refers to specific aspects of service provisioning, which can be characterized as a new class of specification patterns; b) the specification patterns proposed in the research literature are barely used in industrial settings.

The outcome of the study described in [8] drove the design of a new formal specification language, with the primary objective to meet the most significant expressiveness requirements emerged from the study[1]. This paper introduces this new language, called SOLOIST (*SpecificatiOn Language fOr servIce compoSitions inTeractions*). The language is based on a many-sorted first-order metric temporal logic, which has been extended with new temporal modalities that support aggregate operators for events occurring in a certain time window. Expressiveness was not the sole requirement in designing this language. We also wanted the language to express specifications that could lead to automatic formal verification. Indeed, we also show that SOLOIST, under certain assumptions, can

[1] The language can be viewed as a profound revision of a previous attempt [3], driven by the feedback from the field study reported in [8].

be translated into linear temporal logic, allowing for its use with established techniques and tools, both for design-time and for run-time verification.

The rest of this paper is structured as follows. In Sect. 2 we describe the requirements elicitation process for the language and discuss some of the issues faced during its design. Section 3 describes some examples of properties, associated with a BPEL process, which can be expressed with the language. Section 4 introduces SOLOIST, its syntax, and its semantics (both informally and formally); it also shows the use of the language to specify the properties presented in Sect. 3. Section 5 illustrates the translation of SOLOIST to linear temporal logic. Section 6 discusses related work and Sect. 7 concludes the paper, providing some hints for future research.

2 Requirements Elicitation and Design of the Language

2.1 Eliciting Language Requirements from Usage of Specification Patterns in SBAs

In [8] some of the authors presented a study on the use of specification patterns in SBAs. The study analyzed the requirements specifications of two sets of case studies. One set consisted of 104 cases extracted from research papers in the area of specification, verification and validation of SBAs published in the last ten years; the other included 100 service interfaces developed by an industrial partner for its service-oriented information system in the last ten years. During the study, each requirement specification was matched against a specification pattern; in total, we analyzed and classified $290 + 625$ requirements specifications from research and industrial data, respectively.

The study classified the requirements specifications according to four classes of property specification patterns. Three of them correspond to the systems of specification patterns proposed by Dwyer et al. [12], by Konrad and Cheng [23], and by Gruhn and Laue [16]; these patterns have been widely used for the specification and verification of concurrent and real-time systems. The fourth group includes patterns that are specific to service provisioning, and have emerged during the study; these new patterns are:

Average response time (S1) is a variant of the bounded response pattern defined in [23] that uses the average operator to aggregate the response time over a certain time window.

Counting the number of events (S2) is used to express common non-functional requirements such as reliability (e.g., "number of errors in a given time window") and throughput (e.g., "number of requests that a client is allowed to submit in a given time window").

Average number of events (S3) is a variant of the previous pattern that states the average number of events occurred in a certain time interval within a certain time window, as in "the average number of client requests per hour computed over the daily business hours".

Maximum number of events (S4) is a variant of pattern S3 that aggregates events using the maximum operator.

Absolute time (S5) indicates events that should occur at a time that satisfies an absolute time constraint, as in "if the booking is done in the first week of March, a discount is given".

Unbounded elapsed time (S6) indicates the time elapsed since the last occurrence of a certain event.

Data-awareness (S7) (inspired by [11,18]) is a pattern denoting properties that refer to the actual data content of messages exchanged between services as in "every ID present in a message cannot appear in any future message".

Summarizing the results of the study, we report that:

- The majority of requirements specifications stated in industrial settings referred to non-functional properties expressed using aggregate operators (e.g., average, count, maximum); more specifically, the combined usage of patterns S1-S3-S4 accounted for the 81.9% of the specifications, with S3 and S4 being the two most used patterns. Similar requirements were found only rarely in the research literature and when so, they were expressed using the non-aggregated versions of the patterns.
- The two most used patterns in research settings were the "response" and the "bounded response" patterns, defined respectively in [12] and [23].
- The usage of specification patterns from the first three groups in the SBAs research literature were similar to existing data available in literature for other domains.
- The specification patterns proposed in the research literature were barely used in industrial settings.
- The usage of pattern S7 was the same in both set of case studies, ranking at the third place.

2.2 Design Choices

The results reported above have deeply influenced the design of SOLOIST. Our main goal has been to design a formal language that is both expressive—to meet the requirements derived from our field study [8]—and suitable for use with automated verification techniques and tools.

Our starting point has been a temporal logic with metrics: this allows us to support the patterns defined in [12,23,16], i.e., the ones prescribing constraints on the order and/or the occurrence of events, possibly with (real-)time information. Note that this subset is enough to express common patterns such as "response" or "bounded response", defined respectively in [12] and [23]. The logic assumes a discrete time domain, with each occurrence of an event denoted by a time-stamp.

As for supporting the *service provisioning* patterns, we made different decisions. First, we decided not to support patterns referring to absolute or elapsed time (patterns S5 and S6), since this would have notably impacted on the complexity of the translation. Moreover, our field study [8] showed that both of them are used in less than 1% of the specifications; given these data, we maintain this decision does not critically affect the expressiveness of the language as well as its reception by practitioners.

Pattern S7 is supported by adding a first-order quantification to the logic, following the approach proposed in [18]. By making the simplifying assumption that domains over which the quantification ranges are finite, the first-order quantification is mere syntactic sugar, which does not impact on the decidability of the language, but helps to improve its readability. The logic is also many-sorted, to support the different types of the messages exchanged among services.

Regarding patterns S3 and S4, which define properties related to the aggregation[2] of events occurred in a certain time interval h within a certain time window K as in "the average number of service invocations per hour over the last 11.5 hours of operation", we run into different possibilities to represent the observation interval h (i.e., one hour in the example) within the time window K (i.e., 11.5 hours in the example) considered to compute the aggregate value. It could be defined either as a fixed window over adjacent, non-overlapping intervals, or as a sliding window over overlapping intervals. The latter interpretation would require also to define a minimal distance corresponding to the shift of the sliding window, which could be either a fixed value, such as a system tick, or a variable value, such as the time-stamp of each event occurrence (meaning that the window slides variably, according to the occurrences of the events). Furthermore, in both interpretations, one has to make a decision on how to deal with time windows whose length is not an exact multiple of the observation interval; in other words, how to consider the tail of the window whose length is less than the one of the observation interval. After consulting with our industrial partner and evaluating its needs, we decided to support the interpretation with adjacent, non-overlapping observation intervals, where tail intervals whose length is shorter then the observation interval are ignored to express pattern S3 but considered to express pattern S4.

Modeling pattern S2 was straightforward, while for pattern S1 we considered its specific use in the context of SBAs. It shall be used to specify the average response time of invocations made to a certain service over a certain time window. Since a service may provide multiple operations, we decided to include the possibility to specify which operations to consider when computing the aggregate response time, as well as the calling points within the workflow of a service composition from which the invocations originate. Moreover, every service invocation in the scope of an instance of pattern S1 is assumed to be synchronous and actually corresponding to a pair of events, the *start* and *end* one. These events corresponds to the start (end) of an invocation in a precise location of the workflow; a start (end) of an invocation to the same operation of a service but from a different location in the workflow is considered a distinct event. Under these premises, we assume that two subsequent occurrences of the same *start* or *end* event may not happen.

[2] Note that patterns S1–S4 express aggregate statistics, without assuming any underlying probabilistic model.

3 Service Compositions and Their Specifications at a Glance

We consider service compositions defined in terms of the BPEL [2] orchestration language. Very briefly, BPEL is a high-level XML-based language for the definition and execution of business processes, defined as workflows that compose external partner services. The definition of a workflow contains a set of variables; the business logic is expressed as a composition of *activities*. The main types of activities are primitives for communicating with other services (*receive*, *invoke*, *reply*, *pick*) and for executing assignments (*assign*) to variables, as well as control-flow structures like *sequence, while, switch* and parallel *flow*s. Advanced control flow structures, like *event*, *fault*, and *compensation handlers* are also available. We assume that each variable defined in a BPEL process is of an XML simple type; variables that can hold a WSDL message or an XML schema element can be represented by flattening their multi-part structure as a sequence of XML simple type variables.

3.1 Examples of Properties of Service Compositions Interactions

Below we list some examples of properties expressed in natural language, which can be used to specify the interactions of a BPEL process. We assume that the process has an integer variable `foo`, an *invoke* activity named *invA* that takes and returns an integer, an *invoke* activity named *invB* with no input or output parameters, three *receive* activities named *recvP*, *recvQ*, and *recvR* and a reply activity *term* that takes no parameters. The detailed workflow structure of the process as well as the other variables are of no interest for the purpose of this section and are omitted for clarity. All properties are under the scope of an implicit universal temporal quantification as in *"In every process run, ..."*.

1. *"At the end of the execution of the activity* invA*, the value of variable* `foo` *should be equal to 42."*
2. *"The execution of activity* recvP *should alternate with the execution of activity* recvQ*, though other activities different from* recvQ *(respectively,* recvP*) can be executed in between."*
3. *"The response time of activity* invB *should not exceed 4 time units."*
4. *"If activity* invB *has been invoked 4 times in the past 16 units, than activity* recvR *will be executed within 32 time units."*
5. *"When activity* term *is executed, the average response time of all the invocations of activity* invB *completed in the past 720 time units should be less than 3 time units."*
6. *"When activity* term *is executed, the average number of invocations, in an interval of 60 time units, of activity* invB *during the past 720 time units should be less than 4".*
7. *"When activity* term *is executed, the maximum number of invocations, in an interval of 60 time units, of activity* invB *during the past 720 time units should be less than 5".*

4 SOLOIST

4.1 Preliminaries

A signature Σ is a tuple $\langle S; F; P\rangle$ where:

- S is a set of sort symbols, i.e., names representing various domains;
- F is a set of pairs $f\colon s_1 \times \ldots \times s_n \to w$ where $n \geq 0$, f is a function symbol, $s_1 \times \ldots \times s_n \to w$ is the type of f, and $s_1, \ldots, s_n, w \in S$;
- P is a set of pairs $p\colon s_1 \times \ldots \times s_n$ where $n \geq 0$, p is predicate symbol, $s_1 \times \ldots \times s_n$ is the type of p, and $s_1, \ldots, s_n \in S$.

The sets S, F, P of Σ are denoted by $\mathit{Sort}(\Sigma), \mathit{Func}(\Sigma), \mathit{Pred}(\Sigma)$. Notice that constants are modeled as nullary functions of the form $c :\to w$.

Let Σ be a signature. For each sort $s \in \mathit{Sort}(\Sigma)$, we assume a set V_s of variables of sort s disjoint from the constants in $\mathit{Func}(\Sigma)$. Also, for each sort $s \in S$, we define the set of terms of sort s by induction:

- a variable $x \in V_s$ of sort s is a term of type s;
- if $f\colon s_1 \times \ldots \times s_n \to w \in \mathit{Func}(\Sigma)$ and $t_1, \ldots, t_n$ are terms of type $s_1, \ldots, s_n$ respectively, than $f(t_1, \ldots, t_n)$ is a term of type w.

An atom has the form $p(t_1, \ldots, t_n)$, with $p(s_1, \ldots, s_n) \in \mathit{Pred}(\Sigma)$ and terms $t_1, \ldots, t_n$ of type $s_1, \ldots, s_n$.

4.2 Syntax

A SOLOIST formula over Σ is defined inductively by:

- if $t_1, \ldots, t_n$ are terms of type $s_1, \ldots, s_n$ and $p(s_1, \ldots, s_n) \in \mathit{Pred}(\Sigma)$ is a predicate symbol, then $p(t_1, \ldots, t_n)$ is a formula;
- if ϕ and ψ are formulae and x is a variable, then $\neg\phi$, $\phi \wedge \psi$, $\exists x\colon \phi$ are formulae;
- if ϕ and ψ are formulae and I is a nonempty interval over $\mathbb{N}$, then $\phi \mathsf{U}_I \psi$ and $\phi \mathsf{S}_I \psi$ are formulae;
- if $n, K \in \mathbb{N}$, $\bowtie \in \{<, \leq, \geq, >, =\}$, ϕ is a formula of the form $p(t_1, \ldots, t_n)$, with $p(s_1, \ldots, s_n) \in \mathit{Pred}(\Sigma)$ and terms $t_1, \ldots, t_n$ of type $s_1, \ldots, s_n$, then $\mathsf{C}^K_{\bowtie n}(\phi)$ is a formula;
- if $n, K, h \in \mathbb{N}$, $\bowtie \in \{<, \leq, \geq, >, =\}$, ϕ is a formula of the form $p(t_1, \ldots, t_n)$, with $p(s_1, \ldots, s_n) \in \mathit{Pred}(\Sigma)$ and terms $t_1, \ldots, t_n$ of type $s_1, \ldots, s_n$, then $\mathsf{V}^{K,h}_{\bowtie n}(\phi)$ and $\mathsf{M}^{K,h}_{\bowtie n}(\phi)$ are formulae;
- if $n, K \in \mathbb{N}$, $\bowtie \in \{<, \leq, \geq, >, =\}$, $\phi_1, \ldots, \phi_m, \psi_1, \ldots, \psi_m$ are formulae of the form $p(t_1, \ldots, t_n)$—with $p(s_1, \ldots, s_n) \in \mathit{Pred}(\Sigma)$ and terms $t_1, \ldots, t_n$ of type $s_1, \ldots, s_n$—where for all $i, 1 \leq i \leq n, \phi_i \neq \psi_i$, then $\mathsf{D}^K_{\bowtie n}\{(\phi_1, \psi_1), \ldots, (\phi_m, \psi_m)\}$ is a formula.

Additional temporal modalities can be defined from the U_I and S_I modalities using the usual conventions. Note that the arguments of modalities $\mathsf{C}, \mathsf{V}, \mathsf{M}, \mathsf{D}$ can only be atoms, i.e., positive literals; this reflects the fact that they represent the occurrences of certain events, which are then aggregated as prescribed by the modality.

4.3 SOLOIST at Work

In this section we show how SOLOIST can be used to specify properties related to the interactions of a service composition described in BPEL.

Let $\mathcal{A}$ be the set of activities defined in a BPEL process[3]; $\mathcal{A} = \mathcal{A}_{start-inv} \cup \mathcal{A}_{end-inv} \cup \mathcal{A}_{recv} \cup \mathcal{A}_{pick} \cup \mathcal{A}_{reply} \cup \mathcal{A}_{hdlr} \cup \mathcal{A}_{other}$ where:

- $\mathcal{A}_{start-inv}$ ($\mathcal{A}_{end-inv}$) is the set of *start* (*end*) events of all *invoke* activities[4];
- $\mathcal{A}_{recv}$ is the set of all *receive* activities;
- $\mathcal{A}_{pick}$ is the set of all *pick* activities;
- $\mathcal{A}_{reply}$ is the set of all *reply* activities;
- $\mathcal{A}_{hdlr}$ is the set of events associated with all kinds of *handlers*;
- $\mathcal{A}_{other}$ is the set of activities that are not an *invoke*, a *receive*, a *pick*, a *reply*, or related to a handler (e.g., an *assign*, a control structure activity).

Let $\mathcal{A}_{msg} = \mathcal{A} \setminus \mathcal{A}_{other}$ be the set of activities that involve a data exchange, i.e., that have either an input message or an output message attached with them. Each $\mu \in \mathcal{A}_{msg}$ has an arity corresponding to the sum of the simple type variables by which its input and output messages can be represented; each $\mu \in \mathcal{A}_{other}$ is nullary.

A signature Σ to specify the interactions of a BPEL process with partner services by means of SOLOIST can be defined as follows:

- S is the set of XML simple types (e.g., integer, character, string);
- F is the set of functions defined by the scripting language used within the process (e.g., XPath functions on integers and strings);
- $P = \mathcal{A}$. A predicate may correspond to the execution of an activity; its arity and type are then those of the corresponding activity. The usage of the equality predicate between terms of the same XML type is also allowed.

Following the definitions in Sect. 4, the variables of a BPEL process are partitioned into various domains V_s, with $s \in \mathit{Sort}(\Sigma)$.

Below we list the translations into SOLOIST of the formulae presented in Sect. 3, each one with the corresponding item number:

1. $\mathsf{G}(\forall x, y\colon \mathit{invA}_{end}(x, y) \rightarrow \mathtt{foo} = 42)$
2. $\mathsf{G}((\mathit{recvP} \rightarrow \neg \mathit{recvP} \mathsf{U}_{(0,\infty)} \mathit{recvQ}) \wedge (\mathit{recvQ} \rightarrow \neg \mathit{recvQ} \mathsf{U}_{(0,\infty)} \mathit{recvP}))$
3. $\mathsf{G}(\mathit{invB}_{start} \rightarrow \mathsf{F}_{[0,4]}\, \mathit{invB}_{end})$
4. $\mathsf{G}(\mathsf{C}^{16}_{=4} \mathit{invB} \rightarrow \mathsf{F}_{[0,32]} \mathit{recvR})$
5. $\mathsf{G}(\mathit{term}_{end} \rightarrow \mathsf{D}^{720}_{\leq 3}(\mathit{invB}_{start}, \mathit{invB}_{end}))$
6. $\mathsf{G}(\mathit{term}_{end} \rightarrow \mathsf{V}^{720,60}_{\leq 4}(\mathit{invB}_{start}))$
7. $\mathsf{G}(\mathit{term}_{end} \rightarrow \mathsf{M}^{720,60}_{\leq 5}(\mathit{invB}_{start}))$

[3] Activities of a BPEL process can be uniquely identified by means of an XPath expression.

[4] A synchronous *invoke* is characterized both by a *start* event and by an *end* event; an asynchronous *invoke* is characterized only by a *start* event.

4.4 Informal Semantics

The informal semantics of SOLOIST is based on a sequence of time-stamped predicates. A predicate corresponds to an event, which models the execution of an activity defined within a service composition; its arguments are the parameters possibly associated with the activity, such as the input message of a service invocation.

The S_I and U_I modalities have the usual meaning in temporal logics ("*Until*" and "*Since*")[5].

The $\mathsf{C}^K_{\bowtie n}(\phi)$ modality, evaluated in a certain time instant, states a bound on the number of occurrences of an event ϕ, counted over a time window K; it expresses pattern S2.

The $\mathsf{V}^{K,h}_{\bowtie n}(\phi)$ modality, evaluated at a certain time instant τ_i, is used to express a bound on the average number (with respect to an observation interval h, open to left and closed to the right) of occurrences of an event ϕ, occurred within a time window K; this corresponds to pattern S3. As discussed in Sect. 2, since K may not be an exact multiple of h, the actual time window over which occurrences of event ϕ are counted is bounded by $\tau_i - \lfloor \frac{K}{h} \rfloor h$ on the left and τ_i on the right; similarly, the number of observation intervals taken into account to compute the average is $\lfloor \frac{K}{h} \rfloor$. Consider, for example, the sequence of events depicted in Fig. 1, where black circles correspond to occurrences of the ϕ event. Assuming $\tau_i = 42$, $K = 35$, and $h = 6$ (values expressed as time units), $\lfloor \frac{K}{h} \rfloor = \lfloor \frac{35}{6} \rfloor = 5$. The evaluation of the formula $\mathsf{V}^{35,6}_{\bowtie n}(\phi)$ at time instant 42 is then $\frac{2+1+2+4+1}{5} \bowtie n$, where the numerator of the fraction to the left of $\bowtie$ is the number of event occurrences in the window bounded by τ_i and $\tau_i - 5h$.

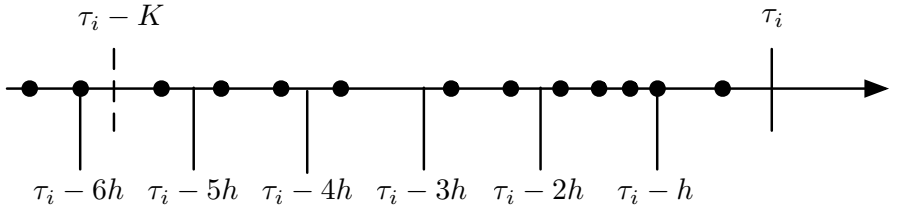

Fig. 1. Sequence of events over a time window K, with observation interval h (semantics of the V and M modalities)

The $\mathsf{M}^{K,h}_{\bowtie n}(\phi)$ modality, evaluated in a certain time instant τ_i, is used to express a bound on the maximum number (with respect to an observation interval h, open to left and closed to the right) of occurrences of an event ϕ, occurred within a time window K; this corresponds to pattern S4. Differently from the V modality described above, this modality takes also into account the events occurring in a tail interval, even if its length is shorter than the one of the

[5] A strict semantics is assumed for the U_I and S_I modalities.

observation interval h. With reference to Fig. 1 and assuming the same values as above for τ_i, K, and h, the tail interval bounded by $\tau_i - K$ on the left and $\tau_i - \lfloor \frac{K}{h} \rfloor h = \tau_i - 5h$ on the right is also considered for computing the aggregate value. This leads to a final evaluation for the formula equivalent to $\max(\{1\} \cup \{4\} \cup \{2\} \cup \{1\} \cup \{2\} \cup \{1\}) \bowtie n = 4 \bowtie n$, where the i-th singleton set in the argument of the aggregate operator corresponds to the number of event occurrences in the i-th observation interval within the time window.

The D modality, evaluated in a certain time window τ_i, expresses a bound on the average time elapsed between pairs of specific adjacent events, occurred within a time window K; it can be used to express pattern S1. Consider, for example, the sequence of events depicted in Fig. 2, where capital letters in the lower part of the timeline correspond to events, and numbers in the upper part of the timeline indicate time-stamps; assume that the current time instant is $\tau_i = 18$ and that $K = 12$. To express a bound for the average distance between each occurrence of an event A and the first subsequent occurrence of an event B, as well as for the pair of events (C, D), for the previous 12 time units, one writes a formula like $\mathsf{D}^{12}_{\bowtie n}\{(A, B), (C, D)\}$, for some $\bowtie$ and n. With respect to $\tau_i = 18$, the time window of length $K = 12$ includes the events (with their respective time-stamp) $(A, 7)$, $(B, 8)$, $(C, 10)$, $(A, 12)$, $(D, 14)$, $(B, 16)$, $(A, 17)$, enclosed in the rectangle in Fig. 2. The average time distance is then computed by summing the differences between the time-stamps of each (A, B) and (C, D) pair (each pair of events is denoted by a different kind of arrow in Fig. 2), and dividing the result for the number of the selected events pairs (3 in the example). Finally, the D modality compares this result with value n, according to the relation defined by $\bowtie$; i.e., the evaluation of $\mathsf{D}^{12}_{\bowtie n}\{(A, B), (C, D)\}$ is $\frac{(8-7)+(16-12)+(14-10)}{3} \bowtie n$. Note that the event $(A, 17)$ is ignored for computing the (average) distance, since it is not matched by a corresponding B event within the selected time window.

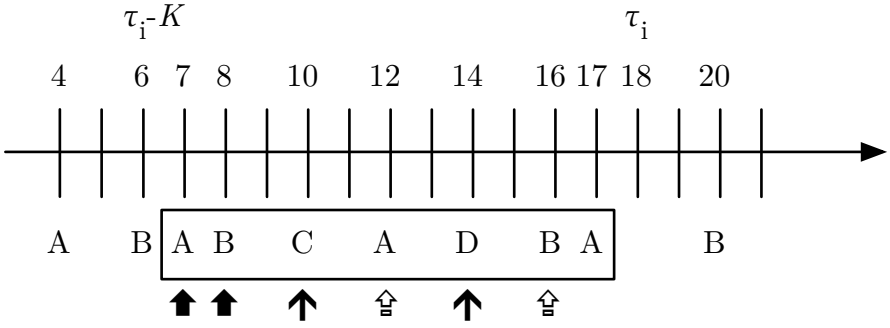

Fig. 2. Sequence of pairs of events over a time window K (semantics of the D modality)

4.5 Formal Semantics

A Σ-structure associates appropriate values to the elements of a signature Σ. A Σ-structure $\mathcal{D}$ consists of:

- a non-empty set $s^{\mathcal{D}}$ for each sort $s \in \mathit{Sort}(\Sigma)$;
- a function $f^{\mathcal{D}} \colon s_1^{\mathcal{D}} \times \ldots \times s_n^{\mathcal{D}} \to w^{\mathcal{D}}$ for each function symbol $f \colon s_1 \times \ldots \times s_n \to w \in \mathit{Func}(\Sigma)$;
- a relation $p^{\mathcal{D}} \subseteq s_1^{\mathcal{D}} \times \ldots \times s_n^{\mathcal{D}}$ for each predicate symbol $p \colon s_1 \times \ldots \times s_n \in \mathit{Pred}(\Sigma)$;

A *temporal first-order* structure over Σ is a pair $(\bar{\mathcal{D}}, \bar{\tau})$, where $\bar{\mathcal{D}} = \mathcal{D}_0, \mathcal{D}_1, \ldots$ is a sequence of Σ-structures and $\bar{\tau} = \tau_0, \tau_1, \ldots$ is a sequence of natural numbers (i.e., time-stamps), where:

- the sequence $\bar{\tau}$ is monotonically increasing (i.e., $\tau_i < \tau_{i+1}$, for all $i \geq 0$);
- for each $\mathcal{D}_i$ in $\bar{\mathcal{D}}$, with $i \geq 0$, for each $s \in \mathit{Sort}(\Sigma)$, $s^{\mathcal{D}_i} = s^{\mathcal{D}_{i+1}}$;
- for each $\mathcal{D}_i$ in $\bar{\mathcal{D}}$, with $i \geq 0$, for each function symbol $f \in \mathit{Func}(\Sigma)$, $f^{\mathcal{D}_i} = f^{\mathcal{D}_{i+1}}$.

A variable assignment σ is a $\mathit{Sort}(\Sigma)$-indexed family of functions $\sigma_s \colon V_s \to s^{\mathcal{D}}$ that maps every variable $x \in V_s$ of sort s to an element $\sigma_s(x) \in s^{\mathcal{D}}$. Notation $\sigma[x/d]$ denotes the variable assignment that maps x to d and maps all other variables as σ does.

The valuation function $[\![t]\!]_\sigma^{\mathcal{D}}$ of term t for a Σ-structure $\mathcal{D}$ is defined inductively as follows:

- if t is a variable $x \in V_s$, then $[\![t]\!]_\sigma^{\mathcal{D}} = \sigma_s(x)$;
- if t is a term $f(t_1, \ldots, t_n)$ then $[\![t]\!]_\sigma^{\mathcal{D}} = f^{\mathcal{D}}([\![t_1]\!]_\sigma^{\mathcal{D}}, \ldots, [\![t_n]\!]_\sigma^{\mathcal{D}})$.

For the sake of readability, we drop the superscript $\mathcal{D}$ and the subscript σ from the valuation function $[\![\cdot]\!]$ when they are clear from the context.

Given a temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ over Σ, a variable assignment σ, symbols $i, n, K, h \in \mathbb{N}$, $\bowtie \in \{<, \leq, \geq, >, =\}$, we define the satisfiability relation $(\bar{\mathcal{D}}, \bar{\tau}, \sigma, i) \models \phi$ for SOLOIST formulae as depicted in Fig. 3.

5 Translation to Linear Temporal Logic

In this section we show how SOLOIST can be translated into linear temporal logic. This translation guarantees the decidability of SOLOIST based on well-known results in temporal logic, allowing for its use with established verification techniques and tools. The translation presented here has not been designed to guarantee efficiency in verification but rather to be comprehensible.

SOLOIST is translated into a variant of linear temporal logic called MPLTL (Metric Linear Temporal Logic with Past) [31], which is a syntactically-sugared version of classical PLTL [21], defined over a mono-infinite discrete model of time represented by ω-words. For simplicity, we assume that the logic underlying SOLOIST is single-sorted; no expressiveness is lost, since it is well-known that

$$
\begin{array}{ll}
(\bar{\mathcal{D}}, \bar{\tau}, \sigma, i) \models p(t_1, \ldots, t_n) & \text{iff } ([\![t_1]\!], \ldots, [\![t_n]\!]) \in p^{\mathcal{D}_i} \\
(\bar{\mathcal{D}}, \bar{\tau}, \sigma, i) \models \neg\phi & \text{iff } (\bar{\mathcal{D}}, \bar{\tau}, \sigma, i) \not\models \phi \\
(\bar{\mathcal{D}}, \bar{\tau}, \sigma, i) \models \phi \wedge \psi & \text{iff } (\bar{\mathcal{D}}, \bar{\tau}, \sigma, i) \models \phi \wedge (\bar{\mathcal{D}}, \bar{\tau}, \sigma, i) \models \psi \\
(\bar{\mathcal{D}}, \bar{\tau}, \sigma, i) \models \exists x\colon \phi & \text{iff } (\bar{\mathcal{D}}, \bar{\tau}, \sigma[x/d], i) \models \phi \\
 & \quad \text{for some } d \in s^{\mathcal{D}} (\text{with } x \text{ of sort } s) \\
(\bar{\mathcal{D}}, \bar{\tau}, \sigma, i) \models \phi \mathsf{S}_I \psi & \text{iff for some } j < i, \tau_i - \tau_j \in I, (\bar{\mathcal{D}}, \bar{\tau}, \sigma, j) \models \psi \\
 & \quad \text{and for all } k, j < k < i, (\bar{\mathcal{D}}, \bar{\tau}, \sigma, k) \models \phi \\
(\bar{\mathcal{D}}, \bar{\tau}, \sigma, i) \models \phi \mathsf{U}_I \psi & \text{iff for some } j > i, \tau_j - \tau_i \in I, (\bar{\mathcal{D}}, \bar{\tau}, \sigma, j) \models \psi \\
 & \quad \text{and for all } k, i < k < j, (\bar{\mathcal{D}}, \bar{\tau}, \sigma, k) \models \phi \\
(\bar{\mathcal{D}}, \bar{\tau}, \sigma, i) \models C^K_{\bowtie n}(\phi) & \text{iff } c(\tau_i - K, \tau_i, \phi) \bowtie n \text{ and } \tau_i \geq K \\
(\bar{\mathcal{D}}, \bar{\tau}, \sigma, i) \models \mathsf{V}^{K,h}_{\bowtie n}(\phi) & \text{iff } \frac{c(\tau_i - \lfloor \frac{K}{h} \rfloor h, \tau_i, \phi)}{\lfloor \frac{K}{h} \rfloor} \bowtie n \text{ and } \tau_i \geq K \\
(\bar{\mathcal{D}}, \bar{\tau}, \sigma, i) \models \mathsf{M}^{K,h}_{\bowtie n}(\phi) & \text{iff } \max \left\{ \bigcup_{m=0}^{\lfloor \frac{K}{h} \rfloor} \{c(lb(m), rb(m), \phi)\} \right\} \bowtie n \\
 & \quad \text{given } lb(m) = \max\{\tau_i - K, \tau_i - (m+1)h\} \\
 & \quad \text{and } rb(m) = \tau_i - mh, \text{ with } \tau_i \geq K \\
(\bar{\mathcal{D}}, \bar{\tau}, \sigma, i) \models \mathsf{D}^K_{\bowtie n}\{(\phi_1, \psi_1), \ldots, (\phi_m, \psi_m)\} & \text{iff } \frac{\sum_{j=1}^{m} \sum_{(s,t) \in d(\phi_j, \psi_j, \tau_i, K)} (\tau_t - \tau_s)}{\sum_{j=1}^{m} |d(\phi_j, \psi_j, \tau_i, K)|} \bowtie n \\
 & \quad \text{with } \tau_i \geq K
\end{array}
$$

where $c(\tau_a, \tau_b, \phi) = |\left\{ s \mid \tau_a < \tau_s \leq \tau_b \text{ and } (\bar{\mathcal{D}}, \bar{\tau}, \sigma, s) \models \phi \right\}|$,
and $d(\phi, \psi, \tau_i, K) =$
$\left\{ (s,t) \mid \tau_i - K < \tau_s \leq \tau_i \text{ and } (\bar{\mathcal{D}}, \bar{\tau}, \sigma, s) \models \phi, t = \min\{u \mid \tau_s < \tau_u \leq \tau_i, (\bar{\mathcal{D}}, \bar{\tau}, \sigma, u) \models \psi\} \right\}$.

Fig. 3. Formal semantics of SOLOIST

many-sorted first-order logic (on which SOLOIST is based) can be reduced to single-sorted first-order logic when the number of sorts is finite. Moreover, since we assume that the domains corresponding to sorts are finite, we can drop the first-order quantification and convert each quantifier into a conjunction or a disjunction of atomic propositions. Similarly, n-ary predicate symbols (with $n \geq 1$) are converted into atomic propositions. For example, a formula of the form $\exists x\colon P(x)$, with x ranging over the finite domain $\{1, 2, 3\}$, is translated into the formula $\bigvee_{x \in \{1,2,3\}} P_x$, where P_1, P_2, P_3 are atomic propositions. We denote with Π the finite set of atomic propositions used in formulae obtained as described above.

These simplifications allow us to replace the temporal first-order structure $(\bar{\mathcal{D}}, \bar{\tau})$ and the variable assignment σ used in the definition of the satisfiability relation of SOLOIST with *timed* ω-words, i.e., ω-words over $2^{\Pi} \times \mathbb{N}$. For a timed ω-word $z = z_0, z_1, \ldots$, every element $z_k = (\sigma_k, \delta_k)$ contains the set σ_k of atomic propositions that are true at the natural time-stamp denoted by $\tau_k = \sum_{i=0}^{k} \delta_i$ (with $\delta_i > 0$ for all $i > 0$). The satisfiability relation for SOLOIST can then be defined over timed ω-words, and it is denoted by $z, i \overset{\tau}{\models} \phi$, with z being a timed ω-word and $i \in \mathbb{N}$; we omit its definition since it can be derived with straightforward transformations from the one illustrated in Fig. 3.

Furthermore, we introduce a normal form where negations may only occur on atoms (see, for example, [31]). First, we extend the syntax of the language

by introducing a dual version for each operator in the original syntax, except for the $\mathsf{C}^K_{\bowtie n}, \mathsf{V}^{K,h}_{\bowtie n}, \mathsf{M}^{K,h}_{\bowtie n}, \mathsf{D}^K_{\bowtie n}$ modalities[6]: the dual of $\wedge$ is $\vee$; the dual of U_I is "*Release*" R_I: $\phi \mathsf{R}_I \psi \equiv \neg(\neg\phi \mathsf{U}_I \neg\psi)$; the dual of S_I is "*Trigger*" T_I: $\phi \mathsf{T}_I \psi \equiv \neg(\neg\phi \mathsf{S}_I \neg\psi)$. For the sake of brevity, we do not explicitly report the semantics of these dual operators; it can be derived straightforwardly from the above definitions. A formula is in *positive normal form* if its alphabet is $\{\wedge, \vee, \mathsf{U}_I, \mathsf{R}_I, \mathsf{S}_I, \mathsf{T}_I, \mathsf{C}^K_{\bowtie n}, \mathsf{V}^{K,h}_{\bowtie n}, \mathsf{M}^{K,h}_{\bowtie n}, \mathsf{D}^K_{\bowtie n}\} \cup \Pi \cup \bar{\Pi}$, where $\bar{\Pi}$ is the set of formulae of the form $\neg p$ for $p \in \Pi$. For the rest of this section, we assume that SOLOIST formulae have been transformed into equivalent formulae in positive normal form.

Under these assumptions, the translation of SOLOIST to MPLTL boils down to expressing the temporal modalities $\mathsf{R}_I, \mathsf{T}_I, \mathsf{U}_I, \mathsf{S}_I, \mathsf{C}^K_{\bowtie n}, \mathsf{V}^{K,h}_{\bowtie n}, \mathsf{M}^{K,h}_{\bowtie n}, \mathsf{D}^K_{\bowtie n}$ in MPLTL, preserving their semantics.

First of all, we should remark that while in the semantics of SOLOIST the temporal information is denoted by a natural time-stamp, in MPLTL the temporal information is implicitly defined by the integer position in an ω-word. However, the model based on timed ω-words and the one based on ω-words can be transformed into each other. Given an ω-word w such that $w, i \models \phi$ (where $w, i \models \phi$ denotes the satisfiability relation over ω-words), it is possible to define a timed ω-word $z = z_0, z_1, \ldots$, with $z_0 = (w_0, 0)$ and $z_k = (w_k, 1)$ for $k > 0$, such that $z, i \overset{\tau}{\models} \phi$. Conversely, given a SOLOIST timed ω-word z, we need to pinpoint in an MPLTL ω-word w the positions that correspond to time-stamps in the z timed ω-word where an event occurred. We add to the set Π a special propositional symbol e, which is true in each position corresponding to a "valid" time-stamp in the z timed ω-word. In the MPLTL semantics, an ω-word w over $\Pi \cup \{e\}$ is defined as follows: $w_k = \sigma_k \cup \{e\}$ whenever τ_k is defined, and $w_k = \emptyset$ otherwise. We then define a mapping ρ from SOLOIST dual normal form formulae into MPLTL formulae, such that we can state that $z, i \overset{\tau}{\models} \phi$ iff $w, \tau_i \models \rho(\phi)$. The mapping ρ is defined by induction as follows:

1. $\rho(p(t_1, \ldots, t_n)) = p(t_1, \ldots, t_n)$.
2. $\rho(\neg p(t_1, \ldots, t_n)) = \neg p(t_1, \ldots, t_n)$.
3. If ϕ and ψ are formulae and x is a variable, then

$$\begin{aligned}
\rho(\phi \wedge \psi) &= \rho(\phi) \wedge \rho(\psi);\\
\rho(\phi \vee \psi) &= \rho(\phi) \vee \rho(\psi);\\
\rho(\exists x\colon \phi) &= \exists x\colon \rho(\phi);\\
\rho(\forall x\colon \phi) &= \forall x\colon \rho(\phi).
\end{aligned}$$

4. If ϕ and ψ are formulae and I is a nonempty interval over $\mathbb{N}$, then

$$\begin{aligned}
\rho(\phi \mathsf{U}_I \psi) &= (\neg e \vee \rho(\phi)) \mathsf{U}_I (e \wedge \rho(\psi));\\
\rho(\phi \mathsf{S}_I \psi) &= (\neg e \vee \rho(\phi)) \mathsf{S}_I (e \wedge \rho(\psi));\\
\rho(\phi \mathsf{R}_I \psi) &= (e \wedge \rho(\phi)) \mathsf{R}_I (\neg e \vee \rho(\psi));\\
\rho(\phi \mathsf{T}_I \psi) &= (e \wedge \rho(\phi)) \mathsf{T}_I (\neg e \vee \rho(\psi)).
\end{aligned}$$

[6] A negation in front of one of the $\mathsf{C}^K_{\bowtie n}, \mathsf{V}^{K,h}_{\bowtie n}, \mathsf{M}^{K,h}_{\bowtie n}, \mathsf{D}^K_{\bowtie n}$ modalities becomes a negation of the relation denoted by the $\bowtie$ symbol, hence no dual version is needed for them.

5. For $\mathsf{C}^K_{\bowtie n}$, we consider only the case $\mathsf{C}^K_{>n}$, since the other possible relations used for $\bowtie$ can be modeled with the following equivalences: $\mathsf{C}^K_{\leq n} \equiv \neg\mathsf{C}^K_{>n}$; $\mathsf{C}^K_{\geq n} \equiv \mathsf{C}^K_{>n-1}$; $\mathsf{C}^K_{<n} \equiv \neg\mathsf{C}^K_{>n-1}$; $\mathsf{C}^K_{=n} \equiv \mathsf{C}^K_{>n-1} \wedge \neg\mathsf{C}^K_{>n}$.

$$\rho(\mathsf{C}^K_{>n}(\phi)) = \bigvee_{0\leq i_1<\ldots<i_{n+1}<K} \left(\mathsf{Y}^{i_1}(e \wedge \phi) \wedge \ldots \wedge \mathsf{Y}^{i_{n+1}}(e \wedge \phi)\right)$$

where the MPLTL modality Y ("*yesterday*") is the past version of "*next*" and refers to the previous time instant. Intuitively, the above MPLTL formula states that in the previous K time instants there have been at least $n+1$ occurrences of the event corresponding to $(e \wedge \phi)$; such a situation satisfies the constraint associated with the original formula defined in SOLOIST.

6. The mapping for the $\mathsf{V}^{K,h}_{\bowtie n}$ modality is defined in terms of the C modality:

$$\rho(\mathsf{V}^{K,h}_{\bowtie n}\phi) = \rho(\mathsf{C}^{\lfloor \frac{K}{h} \rfloor \cdot h}_{\bowtie n \cdot \lfloor \frac{K}{h} \rfloor}\phi)$$

7. For the modality $\mathsf{M}^{K,h}_{\bowtie n}$, we include only the two cases $\mathsf{M}^{K,h}_{<n}$ and $\mathsf{M}^{K,h}_{>n}$, as the others can be derived by properly combining instances of these two:

$$\rho(\mathsf{M}^{K,h}_{<n}\phi) = \left(\bigwedge_{m=0}^{\lfloor \frac{K}{h} \rfloor - 1} \mathsf{Y}^{m\cdot h}\left(\rho\left(\mathsf{C}^h_{<n}\phi\right)\right)\right) \bigwedge \left(\mathsf{Y}^{\lfloor \frac{K}{h} \rfloor \cdot h}\left(\rho\left(\mathsf{C}^{(K \bmod h)}_{<n}\phi\right)\right)\right)$$

$$\rho(\mathsf{M}^{K,h}_{>n}\phi) = \left(\bigvee_{m=0}^{\lfloor \frac{K}{h} \rfloor - 1} \mathsf{Y}^{m\cdot h}\left(\rho\left(\mathsf{C}^h_{>n}\phi\right)\right)\right) \bigvee \left(\mathsf{Y}^{\lfloor \frac{K}{h} \rfloor \cdot h}\left(\rho\left(\mathsf{C}^{(K \bmod h)}_{>n}\phi\right)\right)\right)$$

The formulae above decompose the computation of the maximum number of occurrences of the event $(e \wedge \phi)$ by suitably combining constraints on the number of occurrences of the event in each observation interval within the time window.

8. For the $\mathsf{D}^K_{\bowtie n}$ modality, $\rho(\mathsf{D}^K_{\bowtie n}(\phi, \psi))$ is defined[7] as follows:

$$\bigvee_{0<h\leq\lfloor \frac{K}{2} \rfloor} \left(\bigvee_{\substack{0\leq i_1<j_1<\ldots i_h<j_h<K \\ \text{and} \\ \left(\sum_{m=1}^{h} \frac{j_m - i_m}{h}\right) \bowtie n}} \left(\begin{array}{c} \mathsf{Y}^{i_1}(e \wedge \phi) \wedge \mathsf{Y}^{j_1}(e \wedge \psi) \wedge \\ \ldots \\ \wedge \mathsf{Y}^{i_h}(e \wedge \phi) \wedge \mathsf{Y}^{j_h}(e \wedge \psi) \wedge \\ \neg \left(\bigvee_{\substack{0\leq s<t<K \\ s\notin\{i_1,\ldots,i_h\} \\ t\notin\{j_1,\ldots,j_h\}}} \left(\mathsf{Y}^s(e \wedge \phi) \wedge \mathsf{Y}^t(e \wedge \psi)\right) \right) \end{array} \right) \right)$$

The above formula considers all possible h occurrences (with h up to $\lfloor \frac{K}{2} \rfloor$, as indicated in the outer "or") of pairs of events corresponding to $(e \wedge \phi)$ and $(e \wedge \psi)$. The inner "or" considers a sequence of h pairs of time instants $(i_1, j_1), \ldots (i_h, j_h)$, constrained by the bound represented by $\bowtie n$. The top, right-hand part of the formula imposes that every pair of time instants actually corresponds to the

[7] For the sake of simplicity, we consider the case of only one pair of events (ϕ, ψ), but the formula can be generalized to the case of multiple pairs (ϕ_i, ψ_i).

occurrence of a pair of events; the bottom, right-hand part excludes the case that some pairs of events may occur at time instants which are not in the above sequence.

The complexity of a formula resulting from the translation may be exponential in the size of the constants occurring in the aggregate operators. Without aggregate operators, the translation is linear in the size of the original formula. The only relevant cases for aggregate operators are $\mathsf{C}^K_{>n}$ and $\mathsf{D}^K_{\bowtie n}$, since the other modalities can easily be defined in terms of these two. The mapping for $\mathsf{C}^K_{>n}\phi$ considers all subsets of $n+1$ integers of the set $\{0,\ldots,K-1\}$. Hence, it may require an MPLTL formula of size proportional to $(n+1)\binom{K}{n+1}$, which in the worst case, corresponding to $n+1=\frac{K}{2}$, is $O(K\cdot 2^K)$. The mapping of $\mathsf{D}^K_{\bowtie n}(\phi,\psi)$ essentially requires, in the worst case, to select all possible subsets of set $\{0,\ldots,K-1\}$, i.e., 2^K subsets. Hence, again this may require an MPLTL formula of size $O(K\cdot 2^K)$. As remarked at the beginning of this section, the translation presented above has been designed to show the possibility of reducing SOLOIST to a linear temporal logic; nevertheless, future work will address efficiency in the verification of SOLOIST formulae.

6 Related Work

While performing the field study described in [8], we noticed that the three main formal languages used by researchers in the field of SBAs to specify and verify properties related to service interactions are LTL (Linear Temporal Logic), CTL (Computational Tree Logic), and Event Calculus [24]. While the first two are mainly used to describe untimed temporal relations between events, Event Calculus has been the basis to develop more expressive languages, such as EC-Assertion [27], which can express service guarantees terms such as those captured by patterns S1 and S2. However, it requires to introduce additional constructs in a formula, such as explicit variables to track response time or event counters, as well additional support formulae, like the ones used to maintain a list of variables which are used to compute an aggregate value.

In [8] we also noticed a recurring presence of extensions of temporal logics with support for first-order quantification, namely LTL-FO, CTL-FO [11], LTL-FO+ [17], and CTL-FO+ [18], which enrich the underlying logic to express *data-aware* properties, captured by pattern S7.

In the realm of SBAs there have also been several proposals of languages for specifying service level agreements, mainly targeting quality-of-service (QoS) attributes such as response time and throughput; among them, we mention WSLA [22] and a timeliness-related extension of WS-Agreement [28]. These languages usually do not have any formal or mathematical grounding, but in most cases they define an XML schema containing the definition of the main QoS attributes and their data types. One exception is SLAng, which—besides being defined on the top of standard modeling languages like EMOF and OCL, to guarantee precision and understandability—has been mapped to timed automata, to enable efficient run-time monitoring [34].

The fragment of SOLOIST corresponding to many-sorted metric first-order temporal logic is very similar to the work defined in [6], where a similar fragment is used to define system policies, which are then monitored; however, this fragment, without the other temporal operators introduced in SOLOIST, would have been inadeguate to express the service provisioning patterns.

In the field of (temporal) logics, there have been several proposals to express properties related or similar to the one captured by the service provisioning patterns identified in [8]. For example, references [26] and [25] propose, respectively, Counting CTL and Counting LTL, which extend the temporal modalities of the underlying (non-metric) logic with the ability to constrain the number of states satisfying certain sub-formulae along paths. In [7], a first-order policy specification language is introduced; the language, based on past time linear temporal logic with first-order quantifier, includes also a counting quantifier, used to express that a policy depends on the number of times another policy was satisfied in the past. Rabinovich [33] presents TLC, the metric temporal logic with counting modalities over continuous time, where a counting modality $C_k(X)$ states that X is true at least at k points in the unit interval ahead.

Aggregate operators have been studied in the context of mathematical logic, for database query languages [19] and logic programming [30]. More recently, they have also been considered in temporal logics, to express quantitative atomic assertions related to accumulative values of variables along a computation [9]. de Alfaro [1] introduces an operator to express bounds on the average time between events (conceptually similar to the D operator of SOLOIST) in the context of probabilistic temporal logic, to specify and verify performance and reliability properties of discrete-time probabilistic systems. Extensions of specification formalism with statistical operators have also been proposed in the context of run-time verification. In [13], LTL is extended with operators that evaluate aggregate statistics over an execution trace. Reference [14] presents the LARVA verification tool, based on *Dynamic Automata with Timers and Events*, which is able to evaluate statistical measures over dynamic intervals, like the ones identified with the $\mathsf{C}, \mathsf{V}, \mathsf{M}, \mathsf{D}$ modalities of SOLOIST; however, the report does not provide enough details on the language used to specify the properties to monitor.

7 Conclusion and Future Work

Service-based applications demand rethinking the way software is designed, specified and verified. In this paper we focus on the specification aspect and, in particular, we propose a new language, called SOLOIST, that can be used to specify properties of service compositions interactions. The language has been designed from scratch, after capturing and reasoning on the most common property specification patterns used by practitioners in the field of SBAs. Based on a many-sorted first-order metric temporal logic, SOLOIST includes new temporal modalities that have been tailored to express properties that refer to aggregate operations for events occurring in a certain time window. We also show how SOLOIST can be translated into linear temporal logic, allowing for its use with established techniques and tools for both design-time and run-time verification.

Indeed, our next steps with SOLOIST will focus on its efficient verification based on the Zot toolkit [32], developed[8] within our group, by defining an efficient SMT-based encoding of the language. Although Zot has been used so far for design-time verification, we also want to experiment to embed it and its SOLOIST plug-in within a Web service monitoring architecture (such as Dynamo [15]), to enable support also for run-time verification.

Acknowledgments. This work has been partially supported by the European Community under the the IDEAS-ERC grant agreement no. 227977-SMScom; by the Swiss NSF under the grant agreement no. 135051-CLAVOS; by the National Research Fund, Luxembourg (FNR/P10/03). The authors wish to thank Udi Boker, Srđan Krstić, and Franco Raimondi for their feedback on earlier drafts of this paper.

References

1. de Alfaro, L.: Temporal Logics for the Specification of Performance and Reliability. In: Reischuk, R., Morvan, M. (eds.) STACS 1997. LNCS, vol. 1200, pp. 165–176. Springer, Heidelberg (1997)
2. Andrews, T., et al.: Business Process Execution Language for Web Services, Version 1.1 (2003)
3. Baresi, L., Bianculli, D., Ghezzi, C., Guinea, S., Spoletini, P.: Validation of web service compositions. IET Softw. 1(6), 219–232 (2007)
4. Baresi, L., Di Nitto, E. (eds.): Test and Analysis of Web Services. Springer (2007)
5. Baresi, L., Di Nitto, E., Ghezzi, C.: Toward open-world software: Issue and challenges. IEEE Computer 39(10), 36–43 (2006)
6. Basin, D., Klaedtke, F., Müller, S.: Policy Monitoring in First-Order Temporal Logic. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 1–18. Springer, Heidelberg (2010)
7. Bauer, A., Goré, R., Tiu, A.: A First-Order Policy Language for History-Based Transaction Monitoring. In: Leucker, M., Morgan, C. (eds.) ICTAC 2009. LNCS, vol. 5684, pp. 96–111. Springer, Heidelberg (2009)
8. Bianculli, D., Ghezzi, C., Pautasso, C., Senti, P.: Specification patterns from research to industry: a case study in service-based applications. In: Proc. of ICSE 2012, pp. 968–976. IEEE Computer Society (2012)
9. Boker, U., Chatterjee, K., Henzinger, T.A., Kupferman, O.: Temporal specifications with accumulative values. In: Proc. of LICS 2011, pp. 43–52. IEEE Computer Society (2011)
10. Canfora, G., Di Penta, M.: Service-Oriented Architectures Testing: A Survey. In: De Lucia, A., Ferrucci, F. (eds.) ISSSE 2006-2008. LNCS, vol. 5413, pp. 78–105. Springer, Heidelberg (2009)
11. Deutsch, A., Sui, L., Vianu, V.: Specification and verification of data-driven web applications. J. Comput. Syst. Sci. 73(3), 442–474 (2007)
12. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Property specification patterns for finite-state verification. In: Proc. of FMSP 1998, pp. 7–15. ACM (1998)
13. Finkbeiner, B., Sankaranarayanan, S., Sipma, H.: Collecting statistics over runtime executions. Formal Methods in System Design 27, 253–274 (2005)

[8] `http://code.google.com/p/zot/`

14. Gauci, A., Pace, G.J., Colombo, C.: Statistics and runtime verification. Tech. rep., University of Malta (2010)
15. Ghezzi, C., Guinea, S.: Run-time monitoring in service-oriented architectures. In: Baresi, Di Nitto [4] pp. 237–264
16. Gruhn, V., Laue, R.: Patterns for timed property specifications. Electron. Notes Theor. Comput. Sci. 153(2), 117–133 (2006)
17. Hallé, S., Villemaire, R.: Runtime monitoring of message-based workflows with data. In: Proc. of EDOC 2008, pp. 63–72. IEEE Computer Society (2008)
18. Hallé, S., Villemaire, R., Cherkaoui, O.: Specifying and validating data-aware temporal web service properties. IEEE Trans. Softw. Eng. 35(5), 669–683 (2009)
19. Hella, L., Libkin, L., Nurmonen, J., Wong, L.: Logics with aggregate operators. J. ACM 48, 880–907 (2001)
20. Josuttis, N.: SOA in Practice: The Art of Distributed System Design. O'Reilly Media, Inc. (2007)
21. Kamp, H.W.: Tense Logic and the Theory of Linear Order. PhD thesis, University of California at Los Angeles, USA (1968)
22. Keller, A., Ludwig, H.: The WSLA framework: specifying and monitoring service level agreement for web services. J. Netw. Syst. Manage. 11(1) (2003)
23. Konrad, S., Cheng, B.H.C.: Real-time specification patterns. In: Proc. of ICSE 2005, pp. 372–381. ACM (2005)
24. Kowalski, R., Sergot, M.: A logic-based calculus of events. New Gen. Comput. 4, 67–95 (1986)
25. Laroussinie, F., Meyer, A., Petonnet, E.: Counting LTL. In: Proc. of TIME 2010, pp. 51–58. IEEE (2010)
26. Laroussinie, F., Meyer, A., Petonnet, E.: Counting CTL. In: Ong, L. (ed.) FOSSACS 2010. LNCS, vol. 6014, pp. 206–220. Springer, Heidelberg (2010)
27. Mahbub, K., Spanoudakis, G.: Monitoring WS-Agreements: An event calculus-based approach. In: Baresi, Di Nitto [4], pp. 265–306
28. Müller, C., Martín-Díaz, O., Ruiz-Cortés, A., Resinas, M., Fernández, P.: Improving Temporal-Awareness of WS-Agreement. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 193–206. Springer, Heidelberg (2007)
29. Papazoglou, M.P.: The Challenges of Service Evolution. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 1–15. Springer, Heidelberg (2008)
30. Pelov, N., Denecker, M., Bruynooghe, M.: Well-founded and stable semantics of logic programs with aggregates. Theory and Practice of Logic Programming 7(3), 301–353 (2007)
31. Pradella, M., Morzenti, A., San Pietro, P.: The symmetry of the past and of the future: bi-infinite time in the verification of temporal properties. In: Proc. of ESEC-FSE 2007, pp. 312–320. ACM (2007)
32. Pradella, M., Morzenti, A., San Pietro, P.: A Metric Encoding for Bounded Model Checking. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009. LNCS, vol. 5850, pp. 741–756. Springer, Heidelberg (2009)
33. Rabinovich, A.: Complexity of Metric Temporal Logics with Counting and the Pnueli Modalities. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 93–108. Springer, Heidelberg (2008)
34. Raimondi, F., Skene, J., Emmerich, W.: Efficient online monitoring of web-service slas. In: Proc. of SIGSOFT 2008/FSE-16, pp. 170–180. ACM, New York (2008)
35. Salaün, G.: Analysis and verification of service interaction protocols - a brief survey. In: Proc. of TAV-WEB 2010. EPTCS, vol. 35, pp. 75–86 (2010)