GSM+T: A Timed Artifact-Centric Process Model

Julius Köpke
Alpen-Adria-Universität, Klagenfurt, Austria
julius.koepke@aau.at
https://orcid.org/0000-0002-6678-5731

Johann Eder
Alpen-Adria-Universität, Klagenfurt, Austria
johann.eder@aau.at
https://orcid.org/0000-0001-6050-468X

Jianwen Su
Department of Computer Science, UC Santa Barbara, USA
su@cs.ucsb.edu
https://orcid.org/0000-0002-4637-1339

Abstract

We introduce an extension to the declarative and artifact-centric Guard Stage Milestone (GSM) process modeling language to represent temporal aspects (duration, deadlines, lower- and upper-bound constraints), define the correctness of executions of GSM processes with respect to temporal constraints, check controllability of processes, compute execution plans respecting temporal constraints, and provide a translation method allowing to execute controllable GSM+T processes on standard GSM Engines.

2012 ACM Subject Classification Applied computing → Business process modeling

Keywords and phrases Guard Stage Milestone, GSM, Time Constraints, Controllability, Case Handling, Business Process Modeling

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.18

1 Introduction

Expressing temporal constraints and checking whether a process specification can be executed without violating any constraint (controllability) has been studied for activity-centric processes models for quite some time with works such as [11, 7, 16]. Surprisingly, temporal constraints have not yet been addressed for artifact-centric processes models [15, 9, 1, 13].

We propose to enhance the well-known declarative and artifact-centric process model Guard Stage Milestone (GSM) [15, 9] with temporal constraints and develop techniques for checking their controllability and compute schedules for their correct execution. Since GSM has heavily influenced the new case handling standard CMMN[1] our results also build a foundation for future studies analyzing temporal constraints in CMMN models.

Different notions of correctness of time-constrained processes have been developed, in particular consistency and controllability [7, 8]. Consistency requires that for each possible run there is an execution plan that obeys all temporal constraints. This is generally considered

[1] `http://www.omg.org/spec/CMMN/`

too weak. Controllability requires an execution plan that obeys all temporal constraints for every potential duration of activities and for all possible constellations.

In this paper, we provide the following contributions:

(1) We extend the GSM model with time and temporal constraints (GSM+T).
(2) We formally define the semantics of theses extensions.
(3) We define controllability of GSM+T and provide complete and sound algorithms to check controllability and to compute schedules.
(4) We discuss approaches for the correct execution of GSM+T processes on available GSM engines.

To the best of our knowledge, this is the first approach to consider temporal constraints in artifact-centric process models. In activity centric systems controllability is either checked by constructing schedules along the control structures of process graphs (e.g. [11]) or by mapping timed process models to advanced temporal constraint networks (TCN) (e.g. [7, 17]). Both approaches are not easily applicable for GSM, as it neither has the process graph structure required for the first approach nor do current TCNs support the constructs of GSM.

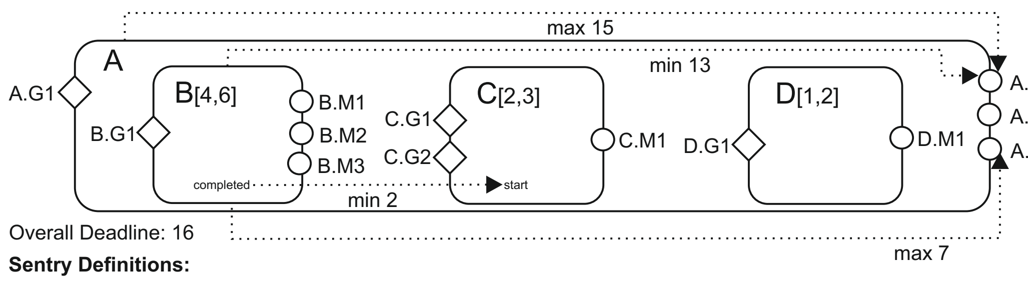
Controllability is a strong condition for correct executability. The work presented here is also a stepping stone for studying more relaxed conditions like dynamic and history dependent controllability [8] for GSM+T.

2 GSM and GSM+T

2.1 Guard Stage Milestone (GSM)

We introduce the relevant aspects of Guard Stage Milestone (GSM) here and refer the reader to [9, 15] for all details. In GSM, a business process is modeled in form of potentially multiple interacting artifacts (or business objects). A process model defines artifact types with a data schema and a life-cycle schema. A life-cycle model consists of *guards*, *stages*, and *milestones*. Guards define conditions under which a stage can be opened. Milestones specify when a specific business goal is reached and the corresponding stage is then closed. Stages can be *atomic* or *composite*. An atomic stage contains a service (e.g. a task). Composite stages contain other composite or atomic stages and are used to group tasks semantically. Guards and milestones are attached to a specific stage and are defined in form of *sentries*. A sentry is an expression of the form "*On event if condition*" or "*On event*" or "*if condition*". An event can be an *internal* or *external* event. Internal events are opening or closing events of stages (denoted by $+stageName$ or $-stageName$) or achievement or invalidation events of milestones ($+milestoneName$ or $-milestoneName$). External events are arbitrary events received from the environment or completion events of services (called from atomic stages). The condition clause of a sentry is a boolean expression over the data-schema. The data schema contains boolean status attributes for stages and milestones and data relevant for the business object.

Execution Semantics. GSM models basically provide a skeleton for restricting rule-based systems. When an external event (e.g. arbitrary or a service completion event) arrives, the payload of the event is incorporated into the data schema of the artifact instance, and potential rules (translated from sentries plus GSM invariants) are triggered leading to the production of internal events and the invocation of external services. Internal events may trigger other rules (micro-steps) until no further internal events are generated. Then, the next external event is picked from an event queue. One such step of consuming an external



Sentry Definitions:

A.G1: on *e1* if *c1*
B.G1: on +A
B.M1: on B.i.completed if c2
B.M2: on B.i.completed if c3
B.M3: on B.i.completed if c4

C.G1: on +B.M1
C.G2: on +B.M2
C.M1: on C.i.completed
D.G1: on C.M1.completed if B.M1
D.M1: on D.i.completed

A.M1: on +D.M1
A.M2: on C.M1 if B.M2
A.M3: on +B.M3

Figure 1 Example GSM Process and some temporal constraints (dotted lines).

event is called a B-Step. The state of the artifact before and after one B-Step is called a Snapshot. A run of a GSM system is a specific sequence of snapshots. All potential runs of a GSM system form a (in general infinite) stage transitions system. GSM prohibits internal events from forming an endless loop (toggle-once principle). During execution, guards and milestones are disjoint. Therefore, only one guard can trigger a single activation of a stage and only one milestone of a stage can be reached for a single invocation.

▶ **Example 1.** In Figure 1 an example GSM process is shown using the usual graphical notion, except from the newly introduced temporal constraints (dotted lines) and service duration intervals (square brackets). It consists of a composite stage A and three atomic stages B, C, D. Stages are depicted in rounded boxes, guards as diamonds, milestones as circles. According to the given sentry definitions (shown below the graphical notation), stage A is opened when the external event $e1$ arrives and the boolean condition $c1$ over the data schema holds. Stage B is opened when A is opened (on $+A$). B is an atomic stage holding the service $B.i$. When $B.i$ completes, one of the milestones $B.M1$ to $B.M3$ is activated. The actual milestone depends on the data conditions $c2$ to $c4$. Stage C is opened when either $B.M1$ or $B.M2$ are reached. C completes when its enclosed service completes. D is opened when $C.M1$ is reached and $B.M1$ was reached. Finally, the milestones of A are reached if either $D.M1$ gets active or $C.M1$ gets active and $B.M2$ is active or if $B.M3$ gets active. Therefore, the process permits the following traces of service executions: $<B>$, $<B, C>$, and $<B, C, D>$. The total number of GSM runs is much higher (potentially infinite) since they also cover all potential data values of the data schema.

Adding Temporal Constraints. In GSM, actual work is performed in atomic stages using external services. Therefore, we specify duration intervals indicating the minimum and maximum execution times of atomic stages. In the graphical representation in Figure 1, we denote the min and max execution time of services within atomic stages in square brackets after the stage name: B takes between 4 and 6 time units (TUs), C between 2 to 3, and D between 1 and 2 TUs. In contrast to activity-centric processes where temporal constraints are defined between activities (tasks), we have selected a more flexible approach for GSM+T: Temporal constraints may be defined between events (external events, opening of stages, service invocations or completions, reaching of milestones). Constraints between external events are typically interpreted as descriptive and constraints involving other events as prescriptive.

The process in Figure 1 includes two upper- and two lower-bound constraints. The time between opening A and reaching milestone $A.M1$ is restricted to max. 15 TUs. E.g.: After a test result has been received the treatment of a patient has to be finished in at most 15 TUs. The time to achieve $A.M3$ after opening B is limited to max. 7 TUs. The time between the completion of the service in stage B and starting the service of stage C must be at least 2 TUs. As an example: Within two days after receiving some medication B a patient must not ingest some medication C. The time between opening B and reaching $A.M1$ must be at least 13 TU. Finally, the example process has an overall deadline of 16 TUs.

2.2 Introducing GSM+T

We have already informally introduced temporal constraints in GSM+T models. In this section, we formalize the GSM+T model.

▶ **Definition 2** (GSM+T Process Model). $(A, X, S, M, G, Y, I, C, B^u, B^l, \Omega)$ is a *GSM+T* process model, if A is a set of artifact types, X is a set of events, S is a set of stages, M is a set of milestones, G is a set of guards, Y is a set of sentries, I is a set of external service invocations, and C is a set of service completions. B^u is a set of upper-bound constraints of the form (s, d, δ), B^l is a set of lower-bound constraints (s, d, δ), where $s, d \in X$ and $\delta \in \mathbb{N}$, and the deadline $\Omega \in \mathbb{N}$.

Events. The set of events X includes a distinct event for each milestone, each stage opening, service invocation and completion of each external service and some external events X^E. $S \cup M \cup I \cup C \cup X^E = X$. Each external event x in X^E has the attribute $x.d_{from}$, $x.d_{to}$ indicating the time interval in which the event may occur after starting the process.

Artifact Types. An artifact type is a tuple (Att, S'), where $Att = Att_{status} \cup Att_{data}$, $S' \subseteq S$. Att_{status} is a set of boolean status attributes for all stages in S' and all milestones of stages in S' indicating currently opened stages and reached milestones of artifact instances. Att_{data} is a set of arbitrary data attributes.

Stages. A stage s is a tuple $s = (G, M, sub, sup, I, C, d_{min}, d_{max})$, where $s.G$ is a non-empty set of guards ($s.G \subseteq G$ and $s.G \neq \emptyset$), $s.M$ is a non-empty set of milestones ($s.M \subseteq M$ and $s.M \neq \emptyset$). The attribute $s.sub$ holds the sub-stages of s ($s.sub \subseteq S$). The attribute $s.sup$ refers to the super stage of s or to s for root stages. Either $s.sub$ or $s.I$ are empty, but never both. $s.I$ refers to an external service in I, $s.C$ refers to the completion event of the external service. Invocation and completion events have an attribute $.S$ referring to their atomic stage. $s.I.d_{min}, s.I.d_{max} \in \mathbb{N}$ are the minimum and maximum duration of the external service, with $0 \leq s.I.d_{min} \leq s.I.d_{max}$.

Sentries of Guards and Milestones. Each milestone $m \in M$ has one sentry $m.se \in Y$. Each guard $g \in G$ has one sentry $g.se \in Y$. Each sentry belongs to exactly one artifact type (Att, S'). A sentry is an expression of the form "on event if condition" where "condition" is optional. A sentry se has two components: $se.trig$ is the triggering event $\in X$ and $se.cond$ is a condition. $se.cond$ has the form $p_s \wedge p_d$, where p_s is a conjunction over positive atoms $\in Att_{status}$ that defines the necessary conditions for the sentry to evaluate to $true$, and p_d is an arbitrary query (over elements in Att_{data}) and negative atoms of p_s. $se.ref$ is the set of all (internal) events $\in X$ that are referenced from p_s via their status attributes. As a short-hand, we use $m.trig$ for $m.se.trig$ and $m.ref$ for $m.se.ref$ for milestones $m \in M$ and $g.trig$ for $g.se.trig$ and $g.ref$ for $g.se.ref$ for guards.

Constraints on GSM+T Models. Milestone belong to exactly one stage: $\forall m \in M\ \exists_1 s \in S : m \in s.M$;

Guards belong to exactly one stage: $\forall g \in G\ \exists_1 s \in S : g \in s.G$;

Invocations and completions belong to exactly one stage: $\forall i \in I\ \exists_1 s \in S : i = s.I$; $\forall c \in C\ \exists_1 s \in S : c = s.C$; Completions belong to the same stage as their invocation: $\forall c \in C\ \forall i \in I\ \exists_1 s \in S : i = s.I \wedge c = s.c$; Stages belong to exactly one artifact type: $\forall s \in S\ \exists_1 a \in A : s \in a.S$; There is exactly one root stage per artifact type: $\forall a \in A\ \exists_1 r \in a.S' : r.sup = r$;
All (recursive) sub-stages of an artifact type are closed under the stages of the artifact type: $\forall a \in A\ \forall s \in a.S' : s.sup \in a.S' \wedge s.sub \subseteq a.S'$.

The remainder of this paper focuses on defining the semantics of the GSM+T and on checking the controllability of GSM+T processes with the following restrictions. In contrast to standard GSM, we restrict sentries to expressions containing triggering events and we require that the status condition p_s is a conjunction of positive atoms while p_d can be any arbitrary predicate, the evaluation of which we can only observe. Requiring conjunctions does not limit the generality of our approach since non-conjunctive terms can always be represented by duplicating the corresponding sentries (guards and milestones) to achieve DNF. Sentries with no triggering events can be replaced by sets of sentries for all possible events. Additionally, we require a somewhat more strict *toggle-once principle* requiring that no guard can trigger the re-activation of a stage after it is executed or a milestone is reached. Active milestones remain active. Therefore, we do not support loops of stages. There is currently no approach for temporal constraints of process models with full support of loops in the literature [5]. Loops are either not supported at all or treated as abstract blocks.

2.3 Semantics of GSM+T

We define the semantics of GSM+T as an extension of the semantics of GSM by (1) timestamping all events of a run and (2) define which conditions these timestamps have to adhere to consider a run as temporally correct. We use a finite abstraction of GSM+T to define time constraints, check controllability and calculating schedules. It is inspired by the polarized dependency graph [15] which is used for defining well-formedness of GSM and it is an extension of timed workflow graphs [11] which are used for defining temporal constrains in activity-centric processes. Basically, the nodes of the graph are the events and guards of the process and edges are defined by dependencies between nodes.

▶ **Definition 3** (Temporal Dependency Graph (TDG)). Let $p = (A, X, S, M, G, Y, I, C, B^u, B^l, \Omega)$ be a GSM+T process. (N, E, B^u, B^l, Ω) is a temporal dependency graph of p, iff: $N = X \cup G$,

$$
\begin{aligned}
E = \quad & \{(n,g) | n \in g.ref \cup g.trig, g \in G \cup M\} && \text{// Referenced events to guards and milestones}\\
& \cup \{(s,i) | s \in S, i = s.I\} && \text{// Stages to service invocations}\\
& \cup \{(i,c) | \exists s \in S : i = s.I \wedge c = s.C\} && \text{// Service invocation to service completion}\\
& \cup \{(s,m) | s \in S, m \in s.M\} && \text{// Stages to their milestones}\\
& \cup \{(g,s) | g \in s.G, s \in S\} && \text{// Guards to their stages}\\
& \cup \{(s,g) | s \in S, g \in s.sub.G\} && \text{// Stages to guards of sub-stages}
\end{aligned}
$$

▶ **Example 4.** Figure 2 shows the TDG of the process in Figure 1. The upper- and lower-bound constraints in B^u and B^l are shown as dotted lines (they are not edges of the TDG). Duration intervals of services are denoted in square brackets.

A scenario is a complete run of a process with timestamps for all elements. We now specify the semantics of the temporal constraints by defining which possible execution scenarios are correct. Then we define schedules as a definition of admissible temporal intervals for the nodes in the TDG.

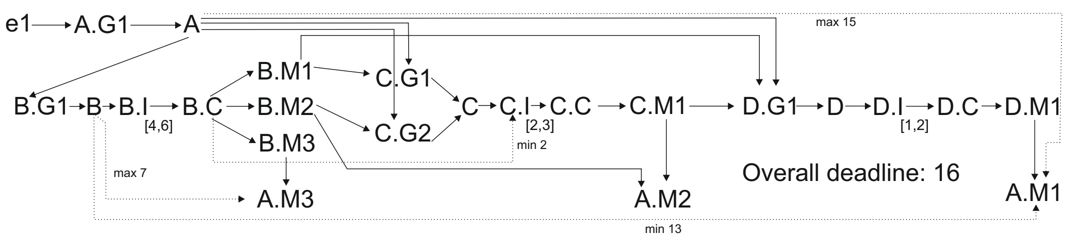


Figure 2 Temporal Dependency Graph of example process in Figure 1.

▶ **Definition 5** (Scenario). A scenario $\bar{S}$ of a GSM+T process assigns timestamps to all elements of a process that occurred during one complete GSM+T run. The TDG ST of a scenario only contains the nodes that occurred during that run. Each node $n \in ST$ is associated with a timestamp $n.t$, the time point of this event in a process instance.

▶ **Definition 6** (Valid Scenario). A scenario $\bar{S}$ of a GSM+T Process (A, X, S, M, G, Y, I, C, B^u, B^l, Ω) with the TDG (N, E, B^u, B^l, Ω), is valid, iff the following constraints hold: $\forall n \in N$, $\forall m \in N$

1. $n \in X^E \Rightarrow n.d_{min} \leq n.t \leq n.d_{max}$
2. $n \in N \wedge (n, m) \in E \Rightarrow n.t \leq m.t$
3. $n \in S \Rightarrow \exists g \in n.G : g.t = n.t$
4. $n = m.trig \Rightarrow m.t = n.t$
5. $n \in S \wedge n.sub = \emptyset \Rightarrow n.I.t + n.I.d_{min} \leq n.C.t \leq n.I.t + n.I.d_{max}$
6. $(s, d, \delta) \in B^u \Rightarrow d.t \leq s.t + \delta$.
7. $(s, d, \delta) \in B^l \Rightarrow s.t + \delta \leq d.t$.

(1) External events occur in their defined intervals. (2) All nodes have equivalent or smaller timestamps than their successors. (3) The guard of a stage has the same time stamp as the stage (In a scenario, there is exactly one guard for each stage.). (4) Nodes with triggering events have the same time point as their triggering event. (5) Service completions of atomic stages occur in their defined intervals. (6-7) upper- and lower-bound constraints are satisfied.

▶ **Example 7.** A fragment (ignoring nodes with equivalent time-stamps) of a valid scenario for the process in Figure 1 is the following: $B.I.t = 0$, $B.I.C.t = 6$, $C.I.t = 8$, $C.I.C.t = 11$, $D.I.t = 12$, $D.I.C.t = 13$. Therefore, $B.I$ directly starts and $B.I.C$ arrives at time 6. $C.I$ is delayed by 2 to obey $lbc(B.I.C, D.I, 2)$ resulting in $C.I.t = 8$. Then $C.I.C$ arrives at time-point 11. $D.I$ is started at time 12 to obey $lbc(B, A.M1, 13)$ and returns at time 13. The scenario complies with all constraints.

2.4 Controllability

Based on these definitions we define the property *controllability* as a notion of the correctness of a process definition with temporal constraints. Controllability reflects the ability of a GSM+T engine to control the execution of GSM+T processes in a way to eventually satisfy all temporal constraints. To define controllability, we have to distinguish four types of events: (1) invocation of a service (task), (2) completion of a service (task), (3) opening/closing of a stage, and (4) achieving a milestone. Only (1) is controlled by the engine, as the engine could delay the invocation of services. (2) depends on external services and (3) and (4) are controlled by conditions and thus automatic. For example, Barcelona [15] is an eager GSM engine, where services are invoked as soon as their enclosing atomic stages are opened. So the only control option for GSM+T execution to satisfy all temporal constraints is the delay of service invocations. Why can such delays be necessary? For an example, if two concurrent stages have to reach their milestones at the same time, then this can only be ensured, if the invocation of the service with shorter duration is delayed.

For simplicity, we assume that each process instance starts at time 0 and all timestamps are relative to the instance start time. The time point of a stage is the time point of opening a stage, thus the time-point of its (activated) guard. The time-point of a guard or milestone is the time-point of the triggering event of its sentry. This is in-line with the GSM semantics [9], where B-Steps/Snapshots have timestamps and micro-steps (triggered by internal events) are assumed to happen at the time of processing the external event triggering the B-Step.

▶ **Definition 8** (Controllability)**.** A GSM+T Process is controllable, iff there exists a mapping T (schedule) which assigns each service invocation $i \in I$ with an execution time $t_i \in \mathbb{N}$ such that for all possible timestamps for external events and for all possible service durations and for all possible guard validations the resulting scenario is valid.

Controllability [21] is a strong condition. More relaxed notions of correctness (e.g. dynamic controllability [8, 16, 21]) would, when adopted to GSM+T, allow the service invocation decisions to depend on already completed snapshots.

3 Checking Controllability and Calculating Schedules

3.1 Schedule Frames and Controllability

In the following we present a procedure for checking the controllability of GSM+T processes by constructing a schedule. Controllability requires that *any* correct schedule exists. We construct schedules which take all the worst cases of durations into account. Faster execution of the process might be possible, as we discuss in Section 4. For computing schedules, we first define a schedule frame of a process as an extension of a TDG:

▶ **Definition 9** (Schedule Frame)**.** A schedule frame associates each $n \in N$ of a TDG (N, E, B^u, B^l, Ω) of a GSM+T process (A, X, S, M, G, Y, I, C, B^u, B^l, Ω) with intervals for the occurrence of nodes: E_b, E_w, L. $n.E_b$ expresses the earliest best time in which n can happen, $n.E_w$ expresses the earliest worst time in which n can happen, $n.L$ is the latest time where n may occur. A schedule frame is correct, iff $\forall n \in N, \forall m \in N$:

1. $n \in X^E \Rightarrow n.E_b = n.d_{min}, n.E_w = n.d_{max}$
2. $n.E_b \leq n.E_w \leq n.L$,
3. $(n,m) \in E \Rightarrow n.E_b \leq m.E_b, n.E_w \leq m.E_w, n.L \leq m.L$
4. $n \in S \wedge n.sub = \emptyset \Rightarrow n.I.E_w + n.I.d_{max} \leq n.C.E_w$
5. $n \in S \wedge n.sub = \emptyset \Rightarrow n.I.E_b + n.I.d_{min} \leq n.C.E_b$
6. $n \in S \wedge n.sub = \emptyset \Rightarrow n.I.L + n.I.d_{max} \leq n.C.L$
7. $n \in S \wedge n.sub = \emptyset \Rightarrow n.E_w \leq n.I.E_b$
8. $n \in G \cup M \Rightarrow n.E_b = n.trig.E_b, n.E_w = n.trig.E_w, n.L = n.trig.L$
9. $\forall (m, m', \delta) \in P.B^u : m'.E_w \leq m.E_b + \delta$,
10. $\forall (m, m', \delta) \in P.B^u : m'.L \leq m.L + \delta$
11. $\forall (m, m', \delta) \in P.B^l : m.E_b + \delta \leq m'.E_w$,
12. $\forall (m, m', \delta) \in P.B^l : m.L + \delta \leq m'.L$
13. $n \in S \Rightarrow n.E_b = min(\{g.E_b | g \in n.G\})$,
    $n \in S \Rightarrow n.E_w = max(\{g.E_w | g \in n.G\})$

(1) The time interval of external events is their defined time interval. (2) Invariant: Earliest best smaller or equal earliest worst smaller or equal latest. (3) Nodes have equivalent or smaller E_b, E_w and L values than their successors. (4-6) The time difference between invocation and completion is defined by the duration interval of the service. (7) The time of an atomic stage is smaller or equal to the time of its service invocation. (8) Guards and milestones have the E_b, E_w, and L values of their triggering events. (9-12) Upper- and lower-bound constraints are satisfied. (13) The earliest best opening time of a stage is the earliest time of all their guards. The earliest worst opening time is the earliest worst time of all their guards.

Example of a schedule frame: The essential parts of a schedule frame - the ones controllable by the engine - for the GSM+T process in Figure 1 in the format (E_b, E_w, L) are the following: *B.I: (0,0,3), C.I: (8,8,11), D.I: (12,12,14)*. The complete schedule frame is shown in the *Backward* 2 column of Table 1.

▶ **Theorem 10.** A GSM+T process P is controllable, iff it has a correct schedule frame.

Proof Sketch. (1) soundness: Let SF be a correct schedule frame for P, let $T(i) = i.E_b$ be a schedule. Following from Def. 9 and Definitions 6, 8 we can show that T fulfills the requirements of controllability.
(2) completeness: Let T be a schedule such that P is controllable. We construct a schedule frame as follows:

1. $\forall i \in I : i.E_b = i.E_w = T(i)$.
2. $\forall s \in S : s.C.E_b = s.I.E_b + d_{min}, s.C.E_w = s.I.E_w + d_{max}$
3. $\forall x \in X^E : x.E_b = x.d_{min}, x.E_w = x.d_{max}$
4. $\forall n \in N - I - C - X^E$:
$n.E_b = max\{m.E_b|(n, m) \in E\}$,
$n.E_w = max\{m.E_w|(n, m) \in E\}$
5. $\forall n \in N : n.L = n.E_w$

With some calculations, it is easy to see that this schedule frame is correct. ◀

▶ **Lemma 11** (Cyclic Dependencies). *If the schedule frame graph contains a cycle the process is not controllable.*

Proof. We distinguish 2 cases: (a) there is an (s, i) edge involved in the cycle. Let S_1 and S_2 be stages contained in a cycle in the dependency graph. This means that both sequences $< S_1, S_2 >$ and $< S_2, S_1 >$ are possible. Hence it is not possible to assign start times to the stages such that both sequences are admissible unless the stages have duration 0.
(b) If there is no (s, i) edge in the cycle, then the process violates the toggle once principle of well-formed GSM. ◀

3.2 Computing Correct Schedule Frames:

We compute a correct schedule frame for an acyclic schedule frame graph of a process P as follows: We initialize the schedule frame as follows: For external event nodes we set the E_b and E_w values to the time intervals of the corresponding events: $(x \in X^E) \rightarrow x.E_b = x.d_{min}, x.E_w = x.d_{max}$. The E-values of all other nodes are set to 0. The L-values are set to the deadline of the whole process ($P.\Omega$). (Computing a schedule is also possible if no overall deadline is given, but the algorithm is considerably longer and out of scope of this paper.)

The events may take place during the interval defined by (E_b, L) of its node. In the course of Algorithm 1 these intervals are now reduced iteratively through incorporating the implicit (an event can only take place after its predecessors) and explicit temporal constraints.

Algorithm 1 CSF(sf,P) Compute correct schedule frame sf for process P.

```
1: {inout: $(N, E, B^u, B^l, d)$ schedule frame, in: $(A, X, S, M, G, Y, I, C, B^u, B^l, \Omega)$ Process}
2: {(N,E) is acyclic}
3: {Output: correct schedule frame or ok is false}
4: $ok := false$
5: **while** $\neg ok$ **do**
6:    $ok := true$
7:    **if** !Forward(($N$, $E$, $B^u$, $B^l$, $d$), ($A$, $X$, $S$, $M$, $G$, $Y$, $I$, $C$, $B^u$, $B^l$, $\Omega$), $ok$) **then**
8:       **return** false
9:    **else if** !Backward(($N$, $E$, $B^u$, $B^l$, $d$), ($A$, $X$, $S$, $M$, $G$, $Y$, $I$, $C$, $B^u$, $B^l$, $\Omega$), $ok$)) **then**
10:      **return** false
11:   **else**
12:      **for all** $(s, d, \delta) \in B^u$ **do**
13:         {incorporation of upper-bound constraints}
14:         **if** $\delta < (d.E_w - s.E_b)$ **then**
15:            $ok := false$
16:            $s.E_b := max(s.E_b, d.E_w - \delta)$
17:            $s.E_w := max(s.E_w, s.E_b)$
18:         **end if**
19:         **if** $\delta < (d.L - s.L)$ **then**
20:            $ok := false$
21:            $d.L := min(d.L, s.L + \delta)$
22:         **end if**
23:      **end for**
24:      **for all** $(s, d, \delta) \in B^u$ **do**
25:         {incorporation of lower-bound constraints}
26:         **if** $\delta > (d.E_b - s.E_w)$ **then**
27:            $ok := false$
28:            $d.E_b := max(d.E_b, s.E_w + \delta)$
29:            $d.E_w := max(d.E_w, d.E_b)$
30:         **end if**
31:         **if** $\delta > (d.L - s.L)$ **then**
32:            $ok := false$
33:            $d.L := min(d.L, s.L + \delta)$
34:         **end if**
35:      **end for**
36:   **end if**
37: **end while**
38: **return** true
```

If a constraint is violated, the E_b and E_w values of nodes are increased and the L values are decreased to the lowest resp. highest value satisfying the constraint. The procedure is repeated until either all constraints are satisfied or the invariant $E_w \leq L$ is violated for any node.

Forward Calculation. Algorithm 2 visits all nodes n in a topological sort order and computes the values $n.E_b$ and $n.E_w$ form the corresponding values of its predecessors according to the definition of correct schedule frames. The calculation depends on the type of the current node:

For completion event nodes, there exists exactly one predecessor, the corresponding service invocation node. The E_b and E_w values are defined by the maximum of the current E_b and E_w values and the E_b and E_w of the service invocation node plus the min/max duration times of the service.

Algorithm 2 Forward(sf,P,ok) Forward calculation.

```
1: {inout: $(N, E, B^u, B^l, \Omega)$ schedule frame, in: $(A, X, S, M, G, Y, I, C, B^u, B^l, \Omega)$ Process, inout: ok Boolean}
2: {(N,E) is acyclic}
3: for all $n \in N - X^E$ in a topological order do
4:    {forward calculation}
5:    if $n \in C$ then
6:       $n.E_b := max(\{n.E_b, n.I.E_b + n.I.d_{min}\})$
7:       $n.E_w := max(\{n.E_w, n.I.E_w + n.I.d_{max}\})$
8:    else if $n \in S$ then
9:       $n.E_b := max(n.E_b, \{m.E_b|(m,n) \in E, m \notin G\}, min\{m.E_b|(m,n) \in E, m \in G\})$
10:      $n.E_w := max(n.E_w, \{m.E_w|(m,n) \in E\})$
11:   else if $n \in I$ then
12:      $n.E_b := max(n.E_b, \{m.E_w|(m,n) \in E\}$
13:      $n.E_w := max(n.E_w, \{m.E_w|(m,n) \in E\}$
14:   else
15:      $n.E_b := max(n.E_b, \{m.E_b|(m,n) \in E\})$
16:      $n.E_w := max(n.E_w, \{m.E_w|(m,n) \in E\})$
17:   end if
18:   if $n \in I$ then
19:      $n.E_b := n.E_w$
20:   end if
21:   if $n \in G \cup M \wedge (n.L \neq n.trig.L \vee n.L \neq n.trig.L)$ then
22:      $ok := false$
23:      $n.L := n.trig.L$
24:      if $n.L < n.E_w \vee n \in X^E \wedge (n.E_b \neq n.d_{min} \vee n.E_w \neq d_{max}$ then
25:         return false
26:      end if
27:   end if
28: end for
29: return true
```

Stage nodes may have multiple predecessors in form of multiple guards and optionally a parent stage. The E_b values of a stage is set to the max of its current E_b value and the E_b value of its parent stage and the minimum E_b value of all guards. I.e. a stage is opened when at least one guard triggers and the parent stage is open. The E_w value is set to the max of the current E_w value and the max E_w of all predecessor nodes. Invocation nodes have exactly one predecessor (a stage). Since we only need to find a controllability schedule, the E_b and E_w value of invocation nodes are set to the maximum of the current value and the E_w value of the stage. The E_b and E_w values of all other node types are defined by the maximum of the current values and the max corresponding values of predecessor nodes.

Algorithm 2 also calculates the value $n.L$ for triggered events, as they must be the same as that of the preceding triggering event. If an update of an L value is required the boolean variable *ok* is set to false which triggers another iteration of forward and backward calculations in Algorithm 1. Algorithm 2 also checks for violations of the $n.E_w \leq n.L$ invariant and returns false if it is violated. A return value of false means that no correct schedule frame exists.

Backward Calculation. Algorithm 3 visits all nodes n in a reverse topological sort order and computes the values $n.L$ from the corresponding values of its successor nodes according to the definition of correct schedule frames. It also calculates the value $n.E_b$, $n.E_w$ for

Algorithm 3 Backward(sf,P,ok) Backward calculation.

```
1: {inout: $(N, E, B^u, B^l, \Omega)$ schedule frame, in: $(A, X, S, M, G, Y, I, C, B^u, B^l, \Omega)$ Process, inout: ok Boolean}
2: for all $n \in N$ in a reverse topological order do
3:   {backward calculation}
4:   if $n \in N - I$ then
5:     $n.L := min(\{n.L\} \cup \{s.L|(n, s) \in E\} \cup \{m.L - \delta|(n, m, \delta) \in B^l\}$
6:   else
7:     {invocation node}
8:     $n.L := min(\{n.L\} \cup \{s.L - s.I.d_{max}|(n, s) \in E\} \cup \{m.L - \delta|(n, m, \delta) \in B^l\})$
9:   end if
10:  if $n \in G \cup M \wedge (n.E_b > n.trig.E_b \vee n.E_w > n.trig.E_w)$ then
11:    {Move wait times along triggering edges}
12:    $ok := false$
13:    $n.trig.E_b := n.E_b$
14:    $n.trig.E_w := n.E_w$
15:  else if $n \in I \wedge (n.E_b < n.S.C.E_b - n.d_{min} \vee n.E_w < n.S.C.E_w - n.d_{max})$ then
16:    {Move wait times from completion to invocation}
17:    $ok := false$
18:    $n.E_b := n.S.C.E_b - n.d_{min}$
19:    $n.E_w := max(n.E_b, n.E_w)$
20:  else if $n \in S$ then
21:    {Move wait times from stage to guards}
22:    for all $e \in \{(g, n) \in E|g \in G \wedge g.E_b < n.E_b)\}$ do
23:      $ok := false$
24:      $d = e.g.E_w - e.g.E_b$
25:      $e.g.E_b := n.E_b$
26:      $e.g.E_w := n.E_b + d$
27:    end for
28:  end if
29:  if $n.L < n.E_w \vee n \in X^E \wedge (n.E_b \neq n.d_{min} \vee n.E_w \neq d_{max})$ then
30:    return false
31:  end if
32: end for
33: return true
```

triggering events as they have to be the same as that of the triggered events. Following the same principle, it increases the $n.E_b$, $n.E_w$ values for invocation nodes and of guards, when required. This basically shifts delays to the corresponding invocations as they are the only times that can be controlled. If any shifting was required the status variable *ok* is set to false. This will later trigger another round of forward and backward calculations in Algorithm 1. During backward calculation, Algorithm 3 also checks for violations of the $E_w \leq L$ invariant and whether the admissible interval for external events had been reduced. It returns false, if any of these invariants is violated. In this case, no correct schedule frame exists and Algorithm 1 also terminates.

▶ **Example 12.** The complete calculation of the schedule frame for the example process in Figure 1 based on the TDG in Figure 2 is shown in Table 1. The result is obtained after two rounds of forward and backward calculations. All data is given in the format (E_b, E_w, L). After the first forward and backward cycle (columns Forward 1, Backward 1), the ubc and lbc constraints are enforced. The ubc $(B, A.M3, 7)$ is violated by $B = (0, 0, 3)$ and $A.M3 = (4, 6, 16)$. In particular, $A.M3.L > B.L + 7$. Therefore, $A.L$ is set to 10.

Table 1 Computing schedule frame for Figure 2.

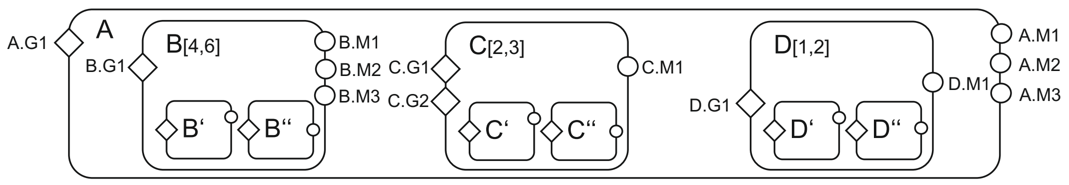
	Forward 1				Backward 1				Forward 2				Backward 2		
Node	E_b	E_w	L		E_b	E_w	L	Node	E_b	E_w	L		E_b	E_w	L
e1	0	0	16		0	0	3	e1	0	0	3		0	0	3
A.G1	0	0	16		0	0	3	A.G1	0	0	3		0	0	3
A	0	0	16		0	0	3	A	0	0	3		0	0	3
B.G1	0	0	16		0	0	3	B.G1	0	0	3		0	0	3
B	0	0	16		0	0	3	B	0	0	3		0	0	3
B.I	0	0	16		0	0	3	B.I	0	0	3		**0**	**0**	3
B.C	4	6	16		4	6	9	B.C	4	6	9		4	6	9
B.M1	4	6	16		4	6	11	B.M1	4	6	9		4	6	9
B.M2	4	6	16		4	6	11	B.M2	4	6	9		4	6	9
B.M3	4	6	16		4	6	11	B.M3	4	6	9		4	6	9
A.M3	4	6	16		4	6	16	A.M3	4	6	9		4	6	9
C.G1	4	6	16		4	6	11	C.G1	4	6	9		4	6	9
C.G2	4	6	16		4	6	11	C.G2	4	6	9		4	6	9
C	4	6	16		4	6	11	C	4	6	11		4	6	11
C.I	6	6	16		6	6	11	C.I	8	8	11		**8**	**8**	11
C.C	8	9	16		8	9	14	C.C	10	11	14		10	11	14
C.M1	8	9	16		8	9	14	C.M1	10	11	14		10	11	14
A.M2	8	9	16		8	9	16	A.M2	10	11	14		10	11	14
D.G1	8	9	16		8	9	14	D.G1	10	11	14		10	11	14
D	8	9	16		8	9	14	D	10	11	14		10	11	14
D.I	9	9	16		9	9	14	D.I	11	11	14		**12**	**12**	14
D.C	10	11	16		10	11	16	D.C	12	13	16		13	13	16
D.M1	10	11	16		10	11	16	D.M1	12	13	16		13	13	16
A.M1	10	11	16		10	11	16	A.M1	13	13	16		13	13	16

The ubc $(A, M1, 15)$ is not violated. The *lbc* constraints $(B, A.M1, 13)$ and $(B.C, C.I)$ are violated. Therefore, the values for $C.I$ and $A.M1$ are updated to $(8, 8, 11)$ and *(13, 13, 16)*, respectively. The change of $A.M1$ is populated to $D.I$ in the following backward calculation phase. One additional iteration of forward and backward calculations is executed without changing values. No additional violations exists and the algorithm terminates. The final results are shown in the column *Backward* 2. A schedule is obtained by only using the E_b values of invocation nodes: *B.I.t: 0, C.I.t = 8, D.I.t = 12.*

3.3 Correctness, Completeness, Complexity, and Feasibility

▶ **Theorem 13.** *A GSM+T process P is controllable iff P is acyclic and Algorithm 1 computes a correct schedule frame.*

Proof. The algorithm terminates always due to the monotonicity of increasing E_b and E_w and decreasing L. Following Theorem 10 we have to show that it computes a correct schedule frame, if there exists a correct schedule frame.
Soundness: All conditions for a correct schedule frame are either explicitly checked, or are immediate result of the calculations and assignments.
Completeness: For all $n \in N$ the values for E_b, E_w, L have to be in the interval $[0, \Omega]$, the variables are initialized accordingly. In the course of the algorithm E_b and E_w can only be increased, L can only be decreased. A change of the value of one of these variables is only done, when a constraint, a condition of a correct schedule frame is violated. Then the variables are set to the lowest resp. highest value which satisfies this constraint. With this monotonicity, we ensure that either a correct set of values is computed or the $E_w \leq L$ invariant is violated or the interval of external events is decreased. ◀



Sentry and Service Definitions:

B‘.G = on +B
B‘.M = on B‘.i.completed
B“.G = on +B‘.M
B“.M = on B“.i.completed

C‘.G = on +C
C‘.M = on C‘.i.completed
C“.G = on +C‘.M
C“.M = on C“.i.completed

D‘.G = on +D
D‘.M = on D‘.i.completed
D“.G = on +D‘.M
D“.M = on D“.i.completed

B‘.i = {complete now}
C‘.i = {complete at time 8}
D‘.i = {complete at time 12}

Figure 3 Standard GSM translation of GSM+T example in Figure 1.

The worst case complexity of the algorithm is $O(n^2)$ where n is the number of nodes in the temporal dependency graph plus the number of explicit constraints. One iteration has n steps and the number of iterations is at least limited by n times the deadline. Preliminary data of explorative experiments with implementations of the basic algorithms, have shown that for reasonably sized models the run-time of the algorithm is within seconds - feasible for design time checks.

4 Execution of GSM+T Processes

In this paper we focus on checking controllability and not on computing optimal schedules nor on the run-time of GSM+T processes. Nevertheless, we discuss in this section how controllable GSM+T processes can be correctly executed to show the feasibility of the GSM+T approach. There are basically two options which rewrite GSM+T processes to be able to run the process on any GSM execution engine.

(1) Fixed execution. Fixed execution has the advantage that there is no need for calculations of time values during the run-time of the process and all invocation times of services are fixed at the process start time. We replace every atomic stage ds by a composite stage ds containing a sequence of two sub-stages ds', which implements a parameterized wait and ds which invokes the requested service. The guards and milestones of ds remain the same but we nest an atomic stage ds' into ds such that ds' has a single guard with the sentry *on* $+ds$. Therefore, it is triggered when ds is opened. The nested stage ds' contains a service call to a special waiting service. The service completion event of $waitDS$ is delayed until the specified time (relative to process start) has passed. Stage ds' has one single milestone $ds'.M$ with the sentry on *ds'.i.completed*. We nest an additional stage ds'' into ds with the guard *on* $+ds'.M$. The service call of ds'' is the service call of the original stage ($ds.impl$). ds'' has a single milestone with the sentry: *on ds”.impl.completed*. The Milestones of ds are not touched. Therefore, they are still connected to the service completion of ds''.

The mapping of the GSM+T model in Figure 1 to a standard GSM Model is shown in Figure 3. According to the schedule (Example 12), the stages B, C, D are replaced by composite stages with wait parameters 0,8, and 12.

(2) Flexible execution. Flexible execution may use the same rewritten process as fixed execution. However, the (possible) delay is computed at run-time by recomputing the schedule frame using the actual values of already executed B-steps and the information on which goals evaluated to true and which milestones were reached. Flexible execution allows for faster execution of the process, but requires significantly more effort at run-time.

5 Related Work

To the best of our knowledge, this is the first approach to consider temporal constraints in artifact-centric process models. For activity-centric process models there is already quite some body of research results in the area of temporal aspects of workflows and business processes. We refer to [12, 4] for overviews. Early approaches for checking process definitions with temporal constraints of activity-centric processes were proposed in works such as [19, 2, 11]. The approaches are based on different techniques such as network analysis, scheduling, or temporal constraint networks [10]. The stream of research based on temporal constraint networks now reached the necessary expressiveness for analyzing business process models: checking the controllability and dynamic controllability of temporal networks, as discussed in [8]. In particular, [16, 6] present sound-and-complete algorithms for checking the dynamic consistency resp. controllability of conditional simple temporal networks with uncertainty.

Works addressing time constrains for interorganizational processes and for service composition were presented in [3, 14]. A quite different approach was proposed in [20], where process mining is used to asses temporal qualities of process models from process logs, but cannot check temporal qualities of process models at build time. Recently the work of Lanz et al. [18] contributed to the consolidation of expressing temporal constraints and defining the semantics of temporal constraints. However, all these works address activity-centric process models and provide no solution for time aware GSM process models.

Our approach is inspired from the algorithms for computing schedules for time constrained workflows presented in [11], but is based on different much more complex type of graph defined to represent the temporal dependencies of GSM+T events and considerably extended to capture the uncertainty of the duration of contingent service calls.

6 Conclusions

Current artifact-centric modeling approaches lack support for temporal constraints. We have extended GSM with deadlines, upper- and lower-bound constraints and intervals defining the potential durations of services. In contrast to activity-centric models, time constraints cannot only be defined between start- and end-times of services but also between (composite) stages, guards and milestones. We have defined controllability of GSM+T models and have provided sound and complete algorithms for checking controllability and calculating schedules. We have provided a method to rewrite controllable GSM+T models to standard GSM models that obey all time constraints when executed on a standard GSM engine (e.g. Barcelona). We see this work as an important foundation for future works including investigating dynamic controllability, where the engine can make different scheduling decisions based on past snapshots and for investigating processes with time dependent sentries.

References

1 Serge Abiteboul, Omar Benjelloun, and Tova Milo. The active xml project: An overview. *The VLDB Journal*, 17(5):1019–1040, 2008. `doi:10.1007/s00778-007-0049-y`.

2 Claudio Bettini, X Sean Wang, and Sushil Jajodia. Temporal reasoning in workflow systems. *Distributed and Parallel Databases*, 11(3):269–306, 2002.

3 J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut. Quality of service for workflows and web service processes. *J. of Web Semantics*, 1(3):281–308, 2004.

4 Saoussen Cheikhrouhou, Slim Kallel, Nawal Guermouche, and Mohamed Jmaiel. The temporal perspective in business process modeling: a survey and research challenges. *Service Oriented Computing and Applications*, 9(1):75–85, 2015.

5 Margareta Ciglic. Time management in workflows with loops. In *Proc. of OTM 2015 Workshops*, pages 5–9, 2015. doi:10.1007/978-3-319-26138-6_2.

6 Alessandro Cimatti, Luke Hunsberger, Andrea Micheli, Roberto Posenato, and Marco Roveri. Dynamic controllability via timed game automata. *Acta Informatica*, pages 1–42, 2016.

7 C. Combi, M. Gozzi, R. Posenato, and G. Pozzi. Conceptual modeling of flexible temporal workflows. *ACM TAAS*, 7(2), 2012. doi:10.1145/2240166.2240169.

8 Carlo Combi and Roberto Posenato. Towards temporal controllabilities for workflow schemata. In *Temporal Representation and Reasoning (TIME), 2010 17th International Symposium on*, pages 129–136. IEEE, 2010.

9 E. Damaggio, R. Hull, and R. Vaculin. On the equivalence of incremental and fixpoint semantics for business artifacts with Guard-Stage-Milestone lifecycles. *Information Systems*, 38:561–584, 2013.

10 Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial intelligence*, 49(1-3), 1991.

11 Johann Eder, Euthimios Panagos, and Michael Rabinovich. Time constraints in workflow systems. In *Advanced information systems engineering*. Springer, 1999.

12 Johann Eder, Euthimios Panagos, and Michael Rabinovich. Workflow time management revisited. In *Seminal Contributions to Information Systems Engineering*. Springer, 2013.

13 Cagdas E. Gerede and Jianwen Su. Specification and verification of artifact behaviors in business process models. In Bernd J. Krämer, Kwei-Jay Lin, and Priya Narasimhan, editors, *ICSOC 2007*, pages 181–192. Springer, 2007.

14 N. Guermouche and C. Godart. Timed model checking based approach for web services analysis. In *ICWS 2009. IEEE*, pages 213–221. IEEE, 2009.

15 R. Hull, E. Damaggio, R. De Masellis, et al. Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In *Proc. of DEBS 2011*, pages 51–62. ACM, 2011.

16 Luke Hunsberger, Roberto Posenato, and Carlo Combi. A sound-and-complete propagation-based algorithm for checking the dynamic consistency of conditional simple temporal networks. In *Temporal Representation and Reasoning (TIME)*. IEEE, 2015.

17 Andreas Lanz, Roberto Posenato, Carlo Combi, and Manfred Reichert. Controllability of time-aware processes at run time. In *On the Move to Meaningful Internet Systems: OTM 2013 Conferences*. Springer, 2013.

18 Andreas Lanz, Manfred Reichert, and Barbara Weber. Process time patterns: A formal foundation. *Information Systems*, 57, 2016.

19 O. Marjanovic and M.E. Orlowska. On modeling and verification of temporal constraints in production workflows. *Knowledge & Inform. Syst.*, 1(2):157–192, 1999.

20 W.M.P. van der Aalst, M.H. Schonenberg, and M. Song. Time prediction based on process mining. *Information Systems*, 36(2):450–475, 2011.

21 T. Vidal and H. Fargier. Contingent durations in temporal csps: From consistency to controllabilities. In *Proc. TIME '97*. IEEE Computer Society, 1997.