On Lifecycle Constraints of Artifact-Centric Workflows*

Esra Kucukoguz and Jianwen Su

Department of Computer Science
University of California at Santa Barbara
{esra,su}@cs.ucsb.edu

Abstract. Data plays a fundamental role in modeling and management of business processes and workflows. Among the recent "data-aware" workflow models, artifact-centric models are particularly interesting. (*Business*) *artifacts* are the key data entities that are used in workflows and can reflect both the business logic and the execution states of a running workflow. The notion of artifacts succinctly captures the fluidity aspect of data during workflow executions. However, much of the technical dimension concerning artifacts in workflows is not well understood. In this paper, we study a key concept of an artifact "lifecycle". In particular, we allow declarative specifications/constraints of artifact lifecycle in the spirit of DecSerFlow, and formulate the notion of lifecycle as the set of all possible paths an artifact can navigate through. We investigate two technical problems: (*Compliance*) does a given workflow (schema) contain only lifecycle allowed by a constraint? And (*automated construction*) from a given lifecycle specification (constraint), is it possible to construct a "compliant" workflow? The study is based on a new formal variant of artifact-centric workflow model called "ArtiNets" and two classes of lifecycle constraints named "regular" and "counting" constraints. We present a range of technical results concerning compliance and automated construction, including: (1) compliance is decidable when workflow is atomic or constraints are regular, (2) for each constraint, we can always construct a workflow that satisfies the constraint, and (3) sufficient conditions where atomic workflows can be constructed.

1 Introduction

Business process management (BPM) has received a rapidly increasing interest in research communities (MIS, CS, and application domains including digital governments, health care delivery, as well as traditional business applications) [5,31]. A key reason is that BPM as a core part of a business enterprise has a wide scope (resource including human management, workflow management, etc.) and is difficult in aspects including discipline barriers among business administration, MIS, IT, etc., and effective management of process/workflow changes. The demand for workflow management tools is enormous. On the other hand, BPM as a research area is rather appealing since there is a lack of a suitable technical framework or model that includes process, data, resources, and human, and can help separating technical problems so that they can be

* Supported in part by NSF grant IIS-0812578 and a grant from IBM.

M. Bravetti and T. Bultan (Eds.): WS-FM 2010, LNCS 6551, pp. 71–85, 2011.

addressed individually and independently [5]. In this paper, we introduce integrity constraints for workflow executions and study the interactions of workflow models and such constraints.

Data plays a fundamental role in modeling and management of business workflows [5]. For example, it is quite often that a workflow starts in response to an (external) request that records some vital information about the request; at each step of a workflow execution, proper bookkeeping is made so that the results of actions taken are reflected and can be used for future decisions, and even the actions themselves are logged for system reasons (e.g., reliability) and business reasons (e.g., accountability). Among the recent proposals for "data-aware" workflow models, artifact-centric models [23,9,2] are particularly interesting. Here (*business*) *artifacts* mean the key data entities that are used in workflows and can reflect both the business logic and the execution states of a running workflow. The notion of artifacts succinctly captures the fluidity aspect of data during workflow executions.

Although artifact-centric modeling approach is suitable for applications [3], many challenges remain in developing technical models for artifact centric workflows. Previous studies on artifact-centric workflow models focused on formal models [9,2,1], verification of temporal properties concerning the workflow logic [9,13,6], static analysis of model well-formedness [2], automated construction from non-temporal goals [8], etc. However, there are still many issues concerning artifacts in workflows that are not well understood.

In this paper, we study the key concept of "lifecycle" for artifacts. A lifespan of an artifact is the sequence of services it encountered during its life time. A lifecycle of an artifact class is the set of all possible lifespan by artifacts in the class. In the formal study, we introduce a variant model for artifact-centric workflows called ArtiNet. ArtiNet workflows resemble Petri nets (see [22] for a tutorial) with two key differences. First, artifacts are used in instead of "tokens". Each artifact belongs to a "class" and places are "typed", i.e., each place may store artifacts of a fixed class. Second, when a transition have multiple input (output) places, only one artifact of each input class is consumed (generated) at each firing.

Motivated by DecSerFlow [32,33], we allow declarative specifications of or constraints on artifact lifecycle. We consider two formalisms for lifecycle specifications: *regular* expressions and semilinear sets of Parikh maps [24] that we call *counting* constraints. As lifecycle constraints, we study the *compliance* problem: does a given workflow only contain lifecycle allowed by a constraint? As lifecycle specification, we investigate the *automated construction* problem: from a given lifecycle specification, is it possible to construct a workflow that "realizes" (satisfies) the specification?

DecSerFlow is a declarative language for specifying permitted sequences of services in a workflow. Earlier work on DecSerFlow focused primarily on implementation of DecSerFlow specifications [33], mapping into SCIFF [21], and verification of logical properties [4]. The compliance problem for regular lifecycle constraints was studied earlier [28,18,17,34]. However, the notion of compliance in these works is "syntactic", i.e., based on containment of transition relations. Our model of compliance is semantic and our results naturally generalize the earlier results.

We present a range of technical results concerning compliance and automated construction problems. We show the following. (1) The compliance problem is decidable for either atomic workflows or regular constraints, with the case of workflows and counting constraints remains open. (2) For each regular/counting specification we can always construct a workflow that realizes the constraint, in particular, each regular constraint is realized by an atomic workflow. (3) We also give cases when atomic workflows can be constructed for counting constraints with or without regular constraints.

This paper is organized as follows. Section 2 motivates the problem with an example. Section 3 defines the ArtiNet model, Section 4 introduces the regular and counting constraints. Sections 5 and 6 focus on compliance and automated construction problems, respectively. Section 7 concludes the paper.

2 Motivations

We use a workflow example to motivate artifact lifecycle constraints and the problems studied in this paper. In this example (Fig. 1), preselling refers to selling an apartment before completing the construction. A preselling permit for a group of apartments must be obtained by a developer before selling activities happen. This workflow is simplified from a similar workflow for real estate management in Hangzhou, China [19] where permits are issued by the real-estate administration office of the city.

In Fig. 1, upon receiving an application, an artifact of PAF (Preselling Approval File) is created. For each apartment listed in the PAF, an AF (Apartment File) is created. Each AF is first verified by a compliance officer, and then evaluated by a pricing estimator. Finally an approving process takes the PAF and the reviews for each AF and approves and completes the application.

As a part of the government requirement, each PAF application must have one preliminary review and and one approving service (task), in that order. This can be easily expressed as a regular constraint "$(\textit{receiving App-Form})(\textit{PAF prelim. review})^{+}(\textit{PAF approving})^{+}$". Another constraint to satisfy involves the number of times services are executed. Note that each apartment in a PAF application need to be reviewed separately. Hence the number of *PAF preliminary review* executions should be the same as the number of *PAF approving* executions, otherwise, some apartments could miss the *PAF approving* service before the entire PAF application is approved. Both example properties are properties on PAF artifact "lifecycle".

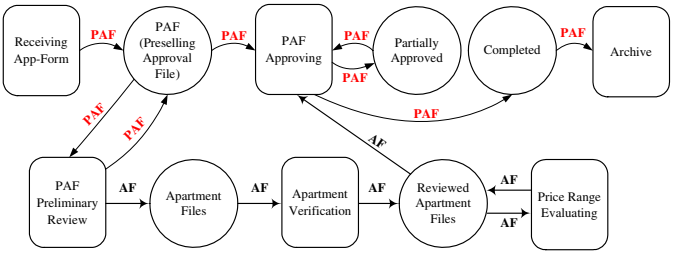


Fig. 1. An ArtiNet Workflow for Apartment Preselling Approval

In this paper we focus on artifact lifecycles, and study the compliance and automated construction problems for workflows in the ArtiNet model. When a new workflow is designed, it is always desirable to know if the workflow is compliant with specified lifecycle constraints. In [28,18,17] authors studied OLCs (object life cycles), and the conformance and coverage problems of OLCs by business process models. A comparison with their results is provided in Section 5. In [34], authors studied the coupling between objects and proposed a method to compute the expected coupling of a process model. Different from these earlier studies, our study also includes automated construction of workflows from lifecycle constraints.

3 ArtiNet: A Formal Model for Artifact-Centric Workflows

In this section, we define the key concepts needed for the technical presentation, including "artifacts", "workflow", "configurations", and "enactments".

Intuitively, artifacts are (abstract) objects that can "flow" through a workflow. The workflow model resembles Petri nets [25] (see e.g., [22]) by substituting tokens with artifacts. Perhaps, the main semantic difference lies in the transition firing rule. But as we will explain below, the firing rule can be simulated by standard Petri nets. Our workflow model is a formal version of ArtiFlow [19] used in the ArtiMT system [20].

In the technical development, we assume the existence of the following countably infinite, pairwise disjoint sets:

- $\mathcal{A}$ of *(artifact) class (names)*,
- $\mathcal{S}$ of services (names),
- $\mathcal{P}$ of *places*, and
- $\mathcal{T}$ of *transitions*.

Definition 1. Let $\boldsymbol{B}$ be a class. A *workflow* for $\boldsymbol{B}$ is a tuple $W = (\Sigma, P, T, \tau_s, \tau_f, E, \boldsymbol{r}, \boldsymbol{s})$ satisfying all of the following conditions:

- Σ is a finite (possibly empty) set of classes such that $\boldsymbol{B} \notin \Sigma$,
- $P \subseteq \mathcal{P}$ is a finite set of places (or repositories),
- $T \subseteq \mathcal{T}$ is a finite set of transitions and $\tau_s, \tau_f \in T$ are the *seed* and *archival* transitions respectively,
- $E \subseteq P \times (T - \{\tau_s\}) \cup (T - \{\tau_f\}) \times P$ is a set of edges,
- $\boldsymbol{r} : P \to \Sigma \cup \{\boldsymbol{B}\}$ is a mapping that assigns each place $p \in P$ a class such that if $(\tau_s, p) \in E$ or $(p, \tau_f) \in E$ then $\boldsymbol{r}(p) = \boldsymbol{B}$,
- For each transition $\tau \in T - \{\tau_s, \tau_f\}$, $(p, \tau) \in E$ and $\boldsymbol{r}(p) = \boldsymbol{B}$ for some place $p \in P$ iff $(\tau, q) \in E$ and $\boldsymbol{r}(q) = \boldsymbol{B}$ for some place $q \in P$, and
- $\boldsymbol{s} : T - \{\tau_s, \tau_f\} \to \mathcal{S}$ is a mapping that assigns each transition a service name.

The workflow W is *atomic* if $\Sigma = \varnothing$.

If W is a workflow for $\boldsymbol{B}$, we also call $\boldsymbol{B}$ the *focused* class of W. (Σ in W consists of *auxiliary* classes.) To simplify the presentation, for the remainder of this paper, we fix some focused class $\boldsymbol{B}$.

Definition 2. Given a workflow $W = (\Sigma, P, T, \tau_s, \tau_f, E, \boldsymbol{r}, \boldsymbol{s})$ for a class $\boldsymbol{B}$, a *configuration* of W is a mapping C from P to natural numbers. A configuration is *empty* if it assigns 0 (zero) to every place.

If a place p is assigned $C(p)$, then p stores $C(p)$ number of artifacts.

Let $W = (\Sigma, P, T, \tau_s, \tau_f, E, \boldsymbol{r}, \boldsymbol{s})$ be a workflow. The set of *input* (resp. *output*) *places* of a transition τ, denoted by $\bullet\tau$ (resp. $\tau\bullet$), is the set of all places that have an edge entering (resp. leaving) the transition τ. We also denote by $(\bullet\tau)_{\boldsymbol{A}}$ (resp. $(\tau\bullet)_{\boldsymbol{A}}$) denote all input (resp. output) places of the transition τ *labeled* $\boldsymbol{A}$. For a set P of places, we define $\boldsymbol{r}(P) = \{\boldsymbol{r}(p) \mid p \in P\}$. In the remaining of this paper, we will frequently use the notations $\boldsymbol{r}(\bullet\tau)$ and $\boldsymbol{r}(\tau\bullet)$ to mean input and output labels of τ, respectively.

Definition 3. Given a workflow $W = (\Sigma, P, T, \tau_s, \tau_f, E, \boldsymbol{r}, \boldsymbol{s})$ for a class $\boldsymbol{B}$, a transition $\tau \in T$ is *enabled* in a configuration C if for each $\boldsymbol{A} \in \boldsymbol{r}(\bullet\tau)$, $\Sigma_{p \in (\bullet\tau)_{\boldsymbol{A}}} C(p) > 0$.

This condition states that a transition is enabled if there is at least one artifact available from each input class from all input places combined. A transition can have multiple input places with the same label, but only one artifact from each input class is sufficient to enable the transition. Note that this is different from standard Petri nets that consume a token from each input place. However, one can simulate this semantics in Petri nets. For example, if a transition τ has input places p, q storing artifacts of $\boldsymbol{B}$. We could construct an additional place p' and two transitions τ_p and τ_q that reads input from p and q (resp.) and outputs to the place p'. We also modify τ's input so that it takes one input from p'. Thus, the "picking an artifact from *either* places" construct in our model is turned into "picking *some* artifact from a place" in Petri nets.

A workflow "moves" from one configuration to another when a transition fires. Given a configuration, there might be more than one possible transitions that are enabled. One of the enabled transitions is chosen to be fired nondeterministically.

When a transition fires, it consumes one artifact from each input class. It generates one artifact for each output class and puts it nondeterministically in one of the output places if there are multiple possible places to put the artifact.

The firing of a transition τ is defined as a triple (C, τ, C'). It indicates transition τ is invoked and the workflow moves from configuration C to configuration C'.

Definition 4. Given a workflow $W = (\Sigma, P, T, \tau_s, \tau_f, E, \boldsymbol{r}, \boldsymbol{s})$, two configurations of C_1, C_2 of W and a transition $\tau \in T$, C_1 *derives* C_2 *using* τ if the following conditions are all true:

1. τ is *enabled* in C_1,
2. If $\boldsymbol{A}$ is an input class and let $p_{\boldsymbol{A}}$ be the place artifact labeled $\boldsymbol{A}$ is chosen to be consumed, then for each input place $p \in (\bullet\tau)_{\boldsymbol{A}}$,

$$C_2(p) = \begin{cases} C_1(p) - 1 & \text{If } p = p_{\boldsymbol{A}} \\ C_1(p) & \text{otherwise} \end{cases}$$

3. If $\boldsymbol{A}$ is an output class and let $p_{\boldsymbol{A}}$ be the place artifact labeled $\boldsymbol{A}$ is put then for each output place $p \in (\tau\bullet)_{\boldsymbol{A}}$,

$$C_2(p) = \begin{cases} C_1(p) + 1 & \text{If } p = p_{\boldsymbol{A}} \\ C_1(p) & \text{otherwise} \end{cases}$$

Definition 5. An *enactment* of a workflow $(\Sigma, P, T, \tau_s, \tau_f, E, \boldsymbol{r}, \boldsymbol{s})$ is an alternating sequence of configurations and transitions $C_0\tau_0C_1\tau_1C_2\cdots\tau_{n-1}C_n$ such that

1. C_0 is the empty configuration,
2. $\tau_0 = \tau_s$ (seed transition) and for all $1 \leqslant i < n$, $\tau_i \in T - \{\tau_s\}$, and
3. For each $0 \leqslant i < n$, C_i derives C_{i+1} using τ_i.

The enactment is *complete* (or *terminating*) if the last configuration C_n is empty and the last transition is the archival transition.

4 Lifecycle Constraints of Artifacts

In this section, we introduce a notion of integrity constraints for ArtiNet workflows, called "lifecycle constraints". Intuitively, lifecycle constraints limits the way how workflow executions should be carried out. In artifact-centric workflows, a workflow enactment (or instance) executes a collection of services in certain order to accomplish the business goal. The constraints in our model thus focus on the sequencing aspect. In the general case, lifecycle constraints could involve the contents of artifacts. However in this initial study, we focus on constraints that do not examine data values. In this section we formulate the key concepts, the technical discussions will be presented in the next two sections.

Clearly, the language DecSerFlow [32,33] for specifying sequencing of service orderings is easily seen as a class of lifecycle constraints. In [29,30], an algebra was developed for event sequences. Algebraic expressions were used as constraints on task sequences during the execution. In [18,28,17] authors studied the consistency of OLC (object life cycle) with respect to a business process model. Finite state machines are used to represent OLCs. In that aspect, our definition of lifecycle coincides with the OLC notion in these papers.

Definition 6. Let $W{=}(\Sigma, P, T, \tau_s, \tau_f, E, \boldsymbol{r}, \boldsymbol{s})$ be a workflow for artifact class $\boldsymbol{B}$ and $C_0\tau_0C_1\tau_1C_2\cdots\tau_{n-1}C_n$ a complete enactment of W. A *lifespan of* $\boldsymbol{B}$ *in* W is a sequence of service names corresponding to the transitions in the complete enactment: $\boldsymbol{s}(\tau_0)\boldsymbol{s}(\tau_1)\cdots\boldsymbol{s}(\tau_{n-1})$. The *lifecycle* $L(W)$ *of* $\boldsymbol{B}$ is the set of all lifespans of $\boldsymbol{B}$ in W.

Note that the seed and archival transitions are ignored in a lifespan. Let W be a workflow. We define $\boldsymbol{S}_W$ as the set of all services that are images of the service mapping in W.

Definition 7. Let W be a workflow for a class $\boldsymbol{B}$. A (*life-cycle*) *constraint on* $\boldsymbol{B}$ is a subset of $\boldsymbol{S}_W^*$, i.e., a (possibly infinite) set of words over $\boldsymbol{S}_W$. The workflow W *satisfies* (or *realizes*) a life-cycle constraint γ, if $L(W) \subseteq \gamma$ (resp. $L(W) = \gamma$).

Let W be a workflow for class $\boldsymbol{B}$. We consider two classes of life-cycle constraints: "regular" and "semi-linear" constraints. A constraint γ on $\boldsymbol{B}$ is *regular* if γ is a regular language over $\boldsymbol{S}_W$. In this paper, we further assume that regular constraints are specified in form of regular expressions [14]. Assuming $a, b, c \ldots$ are services in $\boldsymbol{S}_W$ examples of regular constraints include: $(a + b)^*c$, $(ab^*a)^*$, and $a^*b^*(c + d)$.

To define the second class of constraints, we first assume some fixed enumeration $s_1, ..., s_n$ of $\boldsymbol{S}_W$. The *Parikh map* [24] of a word $w \in \boldsymbol{S}_W^*$ is a vector of natural numbers $\mathit{Parikh}(w) = (v_1, ..., v_n)$ such that for each $i \in [1..n]$, v_i is the number of occurrences of s_i in w. Let $\gamma \subseteq \boldsymbol{S}_W^*$ be a set of words over $\boldsymbol{S}_W$. The *Parikh map* of γ is the set $\mathit{Parikh}(\gamma) = \{\mathit{Parikh}(w) \mid w \in \gamma\}$.

We say that $\gamma \subseteq \boldsymbol{S}_W^*$ is a *counting* constraint if its Parikh map $\mathit{Parikh}(\gamma)$ is a semilinear set [10]. Recall that a linear set is a pair $(\bar{v}, P)$ where $\bar{v}$ is an n-vector and $P = \{\bar{v}_1, ..., \bar{v}_n\}$ is a finite set of *base* vectors. It defines the set of points in n-dimensional space $\{\bar{u} \mid \bar{u} = \bar{v} + \sum_{i=1}^{n} k_i \bar{v}_i$, where $k_i \geqslant 0$ for all $1 \leqslant i \leqslant n\}$. A semilinear set is a finite union of linear sets.

An example of a counting constraint is $i \cdot (1, 2, 0) + j \cdot (0, 0, 2)$ where $i, j \geqslant 0$. This constraint states that s_2 occurs twice as much as s_1 and s_3 occurs an even number of times.

Lemma 4.1. Let W be a workflow for a class $\boldsymbol{B}$ and Γ be a finite set of regular (resp. counting) constraints on $\boldsymbol{B}$. Then there exists a regular (resp. counting) constraint γ such that the following are equivalent:

1. W satisfies every constraint in Γ.
2. W satisfies the constraint γ.

The above lemma easily follows from the fact that both regular languages and semilinear sets are closed under intersection [14,10].

We conclude the section with the following results.

Proposition 4.2. (1) The lifecycle of every atomic workflow is a regular language.
(2) The lifecycle of a workflow is always context-sensitive, and there are workflows whose lifecycles are (i) context-free but not regular, or (ii) not context-free.

Item 1 is rather straightforward. Item 2 is not surprising, since (formal) languages defined by Petri nets are always context-sensitive [26,22].

5 Compliance of Lifecycle Constraints

In this section, we focus on the "compliance" problem for ArtiNet workflows and lifecycle constraints. The problem easily occurs in practice as one would ensure if the existing workflow would satisfy the constraints. We show that the problem is decidable for atomic workflows with either type of constraints, and for workflows with regular constraints. The case of workflows with counting constraints is left open. We also study compliance of DecSerFlow constraints.

Lifecycle Compliance (LC) Problem: Given a workflow W and a regular or counting constraint γ, determine if W satisfies γ.

Unlike the work in [29,30] where enforcement is done at runtime, we study the static analysis problem of deciding if the workflow will always satisfy the constraints. The object lifecycle compliance problem studied in [28,18,17] is closely related to the compliance problem defined above. However, the notion of compliance in [28,18] requires that the object state transitions in the business process is a subset of the transitions in

a specified lifecycle. This is a stronger requirement, i.e., if a workflow is compliant in our model, it may not be compliant by their definition. However, compliance holds in their model would imply compliance in our model. [17] studied compliance considering side-effects while our model does not consider side-effects.

The main results of the section are now stated below.

Theorem 5.1. LC problem is decidable for atomic workflows. For the general case of workflows, LC is decidable for regular constraints.

ArtiNet workflows can be reduced to Petri nets, which as language acceptors, accept context-sensitive languages [22,26]. Since the emptiness problem for context-sensitive languages is undecidable [14], one cannot directly apply known results for the LC problem. On the other hand, Petri nets can be converted to "partially blind multicounter machines" [16], and it is shown [16] that the containment of a Petri net language in a regular language can be determined.

In the remainder of this section, we focus on atomic workflows.

Lemma 5.2. Given an atomic workflow W and a regular constraint γ, it is decidable to check if W satisfies γ.

To establish the lemma, it is sufficient to note that one can effectively convert each regular expression γ to an *equivalent* (non-deterministic) finite state machine M_γ. By Proposition 4.2, the language accepted by W is regular; subsequently, one can construct a finite state machine M_W that accepts the language $L(W)$. It follows that W satisfies γ iff $M_W \subseteq M_\gamma$ (with an abuse of notation). The latter is known to be decidable.

The LC problem for atomic workflows and regular constraints is however PSPACE-complete. This follows from the fact that checking if a non-deterministic FSA accepts Σ^* is PSPACE-hard [11]. We note here that the main source of complexity is from constraints and not workflows. To see its membership in PSPACE, we note that the containment $M_1 \subseteq M_2$ of two FSAs can be reduced to checking the emptiness of $M_1 \cap M_2^c$ (M_2^c is the complement of M_2). Although M_2^c has an exponential size, the emptiness checking can be done without writing down the complete complement machine.

Lemma 5.3. Let γ be a regular constraint that is equivalent to a deterministic FSA M_γ. It can be determined in $O(n^2)$ time if an atomic workflow satisfies γ, where n is the size of the workflow and M_γ.

Clearly, the workflow can be viewed as an FSA M_W. $M_W \subseteq M_\gamma$ iff $M_W \cap M_\gamma^c$ is empty. Since M_γ is deterministic, M_γ^c and M_γ have the same size. Finally, the emptiness of intersection of two FSAs can be determined in the size of their product.

An interesting application of Lemma 5.3 is on compliance of DecSerFlow [32,33] constraints by ArtiNet workflows. DecSerFlow is a declarative language for specifying permitted sequences of tasks in a workflow. DecSerFlow is based on LTL [7] with restrictions in syntax and semantics.

A DecSerFlow specification includes a finite set of services (tasks) and a set of constraints. It defines a set of sequences of services permitted in workflow execution. There are two types of constraints. The first type is cardinality constraints that are associated with individual services, which limits the number of invocations allowed for a service (i.e., occurrences of the service in the sequence). The second type of constraints are

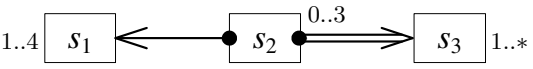
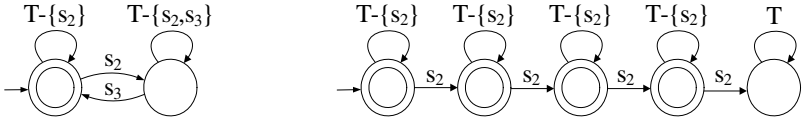


Fig. 2. A DecSerFlow Specification



(a) Alternate response DecSerFlow constraint (b) DecSerFlow cardinality constraint

Fig. 3. FSAs corresponding to DecSerFlow constraints

relational constraints. A relational constraint can be associated with two services and limits occurrences and/or ordering of the involves services.

Fig. 2 shows a simple DecSerFlow specification with three services and two relational constraints. The cardinality constraints restrict the lower and upper bounds of the numbers of occurrences of services ($*$ means unbounded). The relational constraint from s_2 to s_1 requires that each s_2 must be followed by some s_1, i.e., s_2 cannot occur as the last task. The constraint from s_2 to s_3 requires that between two occurrences s_2 there must be an s_3 and that some s_3 must appear after the last s_2.

There are 11 types of relational constraints in DecSerFlow and some of them have negated versions [32,33]. For example, *existence-response* on two services s_1, s_2 means that if s_1 executes, s_2 should also execute but the timing of these executions can be arbitrary; the relational constraint *response* from s_1 to s_2 requires that whenever s_1 executes, there is a "responding" execution of s_2 *later on*. In this case, timing is important. However, s_2 does not have to execute immediately after s_1, and two or more executions of s_1 may share the same s_2 responding execution. The dual relational constraint *precedence* from s_1 to s_2 enforces that if s_2 executes there must be a preceding s_1 execution.

We can easily view each DecSerFlow specification as a lifecycle constraint. In this sense, the following can be shown.

Theorem 5.4. Let γ be a DecSerFlow constraint and W a workflow. It can be decided in $O(nml)$ time if W satisfies γ, where n is the size of W, m the total number of services and relational constraints in γ, and l the largest integer in the cardinality bounds in γ.

In Fig. 3(a), we give the FSA corresponding the *alternate response* constraint in DecSerFlow between s_2 and s_3 showed in Fig. 2. (T stands for the finite set of services.) In this FSA, after each s_2 occurrence, there has to be at least one s_3 before the next s_2 occurrence. Therefore, this FSA is exactly the *alternate response* constraint. In Fig. 3(b), we give the FSA corresponding to the cardinality constraint on s_2 in Fig. 2, where s_2 can occur at most 3 times.

For each relational constraint γ' in γ, we can easily construct a deterministic FSA with a constant number of states. By Lemma 5.3, satisfaction of γ' can be tested in $O(n)$ time. For each cardinality constraint, we can also construct a deterministic DFA with at most l number of states. The theorem follows easily. We now turn to counting constraints.

Lemma 5.5. Given an atomic workflow W and a counting constraint γ, it can be determined if W satisfies γ.

From Proposition 4.2, the lifecycle of W is a regular language. To establish decidability in Lemma 5.5, we note that the Parikh map of each regular language is semilinear [15]. It is also known that semilinear sets are closed under intersection, union, and complement [10]. Thus, W satisfies γ iff *Parikh*$(L(W)) \cap \overline{\gamma}$ is empty ($\overline{\gamma}$ denotes the complement of γ), the latter is decidable [12]. (An alternative is to construct a Presburger formula that states the containment *Parikh*$(L(W)) \subseteq \gamma$; the decidability follows from the decidability of Presburger arithmetic [27].)

6 Workflow Construction from Lifecycle Specification

In this section, we treat a lifecycle constraint as a workflow specification and study the problem of constructing a workflow from a given constraint. It is obvious that one can always construct an atomic workflow for each regular constraint. Therefore, we restrict our attention to (atomic) workflows and counting constraint or in conjunction with a regular constraint, and study the problem of whether from a given counting constraint we can construct an (atomic) workflow. We show that for each counting constraint we can effectively construct a workflow that realizes (satisfies) the constraint. We also exhibit various cases when atomic workflows can be constructed for a counting constraint.

Automated Construction Problem: Given a counting constraint γ, can we find an (atomic) workflow that realizes (satisfies) γ?

Theorem 6.1. (a) For every regular constraint γ, there is an atomic workflow that realizes γ.
(b) For every counting constraint γ, there is a workflow that realizes γ.

Part (a) of Theorem 6.1 is straightforward. In particular, γ can be converted into a nondeterministic FSA M_γ, and then one can construct an atomic workflow by creating a transition node for each transition in M_γ and a place node for each state in M_γ.

Since each DecSerFlow constraint can be expressed as a regular constraint, the following holds.

Corollary 6.2. Each DecSerFlow constraint can be realized by an atomic workflow.

For Part (b), we use auxiliary artifacts to help counting. For one linear set counting constraint, we can construct one workflow that realizes (satisfies) the constraint. If the counting constraint γ is semilinear, i.e., a finite union of linear sets, we can construct a workflow for each linear set and then combine the workflows into a single workflow.

To demonstrate the key idea and techniques, consider a counting constraint $2z_a = z_b$ that states the number of executions of transition b should be twice as much as the number of occurrences of transition a. The Parikh map of the constraint is $\{(i, 2i) \mid i \geqslant 0\}$, i.e., the base vector is $(1, 2)$. To realize this constraint, in addition to our focused artifact ***B*** we also have one auxiliary artifact ***A***. There is one place for class ***B***, all transitions labeled a or b take a ***B*** artifact from and put it back to the place when they are executed. We have 4 places for the auxiliary artifact ***A*** each representing the vectors

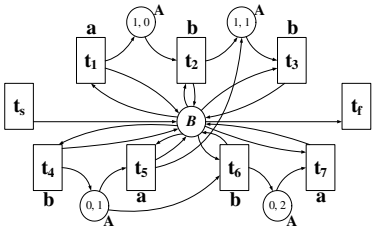


Fig. 4. Realization of a Counting Constraint

(i, j), where $0 \leqslant i \leqslant 1$ and $0 \leqslant j \leqslant 2$, except for $(0, 0)$ (not started yet) and $(1, 2)$ (completed). Each of the a transitions takes an ***A*** artifact from (i, j) for some i, j and after execution puts it into $(i+1, j)$ (or just archive it), therefore simulating the increase in the number of a's. Similarly, each of the b transitions should increase the b counts by 1 while leaving a count unchanged. It is clear that the constraint on the numbers of a's and b's has to be satisfied in complete enactments. Fig. 4 shows the workflow constructed for the counting constraint $2z_a = z_b$. In a complete enactment of this workflow, all the places should be empty at the end. Therefore, all tokens of the auxiliary artifact ***A*** should already be consumed. It can be clearly seen in the figure that when all tokens of ***A*** is consumed, the number of executions of transitions b is exactly the twice as much as the number of executions of transition a.

We now describe the construction of a workflow from a counting constraint. We start from linear sets with a single base vector $(m_1, ..., m_k)$, where k is the number of services. We create place nodes for class ***A*** for each vector $(n_1, ..., n_k)$ where $0 \leqslant n_i \leqslant m_i$ for each $0 \leqslant i \leqslant k$, except the two vectors $(0, ..., 0)$ and $(m_1, ..., m_k)$. For each place p representing the vector $(n_1, ..., n_k)$ where $\exists j, n_j = 1, n_r = 0$ if $r \neq j, 0 \leqslant j, r \leqslant k$, we create an *initialization* transition τ and an edge (τ, p). For each place p representing the vector $(n_1, ..., n_j, ..., n_k)$, if there is a place p' with the vector $(n_1, ..., n_{j-1}, n_j + 1, n_{j+1}, ..., n_k)$, then we create a *continuation* transition τ and two edges (p, τ), and (τ, p'). For each place p representing the vector $(n_1, ..., n_k)$ where $\exists j, n_j = m_j - 1, n_r = m_r$ if $r \neq j, 0 \leqslant j, r \leqslant k$, we create an *ending* transition τ and an edge (p, τ). In fig. 4, created *initialization* transitions are t_1 and t_4, *continuation* transitions are t_2, t_5, t_6, and *ending* transitions are t_3 and t_7. In addition, we create a place node p_* for the focused artifact ***B*** and edges (τ_s, p_*) and (p_*, τ_f). We also create edges (τ, p_*) and (p_*, τ) where $\tau \neq \tau_s, \tau_f$. It can be shown that the Parikh map constraint $\{i \cdot (m_1, ..., m_k) \mid i \geqslant 0\}$ is satisfied. When a linear set has more than one base vector, the construction would repeat for each base vector, except that the common place p_* for the focused class is shared. If the counting constraint γ is a finite union of linear sets, we will construct a workflow with disjoint places for linear set and let the seed transition to (randomly) pick one ***B*** place to start.

In the remainder of the section, we study the problem of constructing atomic workflows: Given a counting constraint γ, find an atomic workflow that satisfies γ.

Clearly, if the γ is a regular language, it reduced to Theorem 6.1(a). We assume that γ is not regular. Actually, the class of counting constraints includes all context-free languages [24]. Naturally, we are looking for atomic workflows whose lifecycles are sublanguages of γ.

Since atomic workflows are basically FSAs, in the following discussions, we focus on finding regular sublanguages called "factors" rather than constructing workflows.

Definition 8. Given two languages L_1 and L_2, L_2 is said to be a *factor* of L_1 if $L_2 \subseteq L_1$, L_2 is infinite and regular.

Notion of factor naturally leads to the following strategy for automated constructions. Given a constraint γ. Let $L_1, ..., L_k$ be factors of γ, we construct a FSA (workflow) for L where $L = \cup_{1 \leqslant i \leqslant k} L_i$.

Lemma 6.3. The language $L \subseteq \{a, b\}^*$ such that *Parikh*$(L) = \{(i, i) \mid i \geqslant 0\}$ has infinitely many pairwise incomparable factors.

To prove the above lemma, we construct the sets E_n's ($n \geqslant 1$) as follows. $E_n = \{w \mid$ *Parikh*$(w) = (n, n)$ and $w \notin \cup_{i=1}^{n-1} E_i^{\oplus}\}$, where $E_i^{\oplus} = \{w^j \mid w \in E_i$ and $j \geqslant 1\}$.

Claim 1: For each $n \geqslant 1$, E_n is not empty.

Claim 1 holds because there are no words in $\cup_{i=1}^{n-1} E_i^{\oplus}$ that starts with n a's. However, $a^n b^n \in E_n$.

Claim 2: For each $i \geqslant 1$, and each $w \in E_i$, $w^* \subseteq L$.

One can easily show that *Parikh*$(w^*) \subseteq \{(i, i) \mid i \geqslant 0\}$; therefore $w^* \subseteq L$.

Claim 3: For each pair of words $w_1, w_2 \in \cup_i E_i$, if $w_1 \neq w_2$ then $w_1^* \nsubseteq w_2^*$.

If we can find a word $w \in w_1^*$ and $w \notin w_2^*$, then $w_1^* \nsubseteq w_2^*$. Consider $w = w_1$.

If $|w_1| < |w_2|$, then there are no words in w_2^* with the same length as w_1, since each word in the set w_2^* has length $|w_2|$ or greater. Therefore, $w_1 \notin w_2^*$.

If $|w_1| > |w_2|$, to satisfy the condition $w_1 \in w_2^*$, we should have $w_1 = w_2^i$ for some $i \geqslant 1$. This violates our construction, hence such w_1 cannot exists.

If $|w_1| = |w_2|$, only word in w_2^* that has the same length with w_1 is w_2. But obviously, $w_1 \neq w_2$, therefore $w_1 \notin w_2^*$.

Lemma 6.3 naturally generalizes to the following.

Theorem 6.4. Let Σ be a (finite) alphabet. Every non-regular counting constraint over Σ^* has infinitely many pairwise incomparable factors.

L has at least one non-regular subset whose Parikh map is a linear set. Let's assume L' is such a subset of L. Since L' is not regular, there is at least one base vector for the Parikh map of L' that defines a non-regular language. The corresponding non-regular constraint can be written as $\{i \cdot (m_a, m_b, \ldots, m_z)\}$ and at least two of the m_is are non-zero. If at most one of them is non-zero, then it defines a regular language. For the basis vector $\{i \cdot (m_a, m_b, \ldots, m_z)\}$, one can come up with infinitely many pairwise incomparable factors, using a similar construction as shown in Lemma 6.3. In fact, these factors can be constructed as: for each $i \geqslant 1$, $(a^{i \cdot m_a} b^{i \cdot m_b} \cdots z^{i \cdot m_z})^*$. Since L' is a subset of L and L' has infinitely many pairwise incomparable factors, so does L.

Theorem 6.4 shows even if the constraints define non-regular languages, "compliant" implementations by atomic workflows are still possible. And in fact, there are many ways to choose from. However, the following result shows that combining regular and counting constraints may sometimes prohibit atomic workflows.

Lemma 6.5. $L = \{a^n b^n \mid n \geqslant 0\}$ has no factors.

Assume L has a factor L'. Let M be the FSA that accepts the language L'. Since L' is infinite, there exists a word $w \in L'$ such that $w = a^i b^i$ and $|w| >$ number of states in M. By the pumping lemma for regular languages, M must include words not in L (thus not in L').

In spite of the above negative result, in the following, we show that it may still be possible to find atomic workflows for a pair of regular and counting constraints.

Lemma 6.6. Let γ_r and γ_c two a regular and counting (resp.) constraint. if γ_r has no union and one star, then $\gamma_r \cap \gamma_c$ defines a regular language.

Intuitively, the above lemma holds since for each linear set, and each union-free regular expression with at most one star, their intersection must be regular. It is because with no stars, the language is finite and so is the intersection. With one star, the Parikh map of the corresponding regular language can be expressed linear equations with the number i of iterations as a parameter. The intersection is thus a linear set and is either finite or infinite including all i's greater than some fixed number.

With this positive result, it is interesting to obtain more sufficient conditions for factor existence in presence of a regular and a counting constraint.

7 Conclusions

In this paper we provide a formal analysis on artifact-centric workflows, centering around the notion of artifact lifecycle. Although our work is inspired by DecSerFlow, the technical problems examined provide new insights into the interplay between constraints/specifications and workflow models. On one hand, compliance problems are solvable in case of workflow and regular expressions (but remain open for counting constraints). On the other hand, construction of workflows can always be done. Our results do not provide a complete characterization of complexity for the technical problems. There are many open problems, including: compliance of counting constraints by workflows, sufficient (and/or necessary) conditions for existence of atomic workflows for counting constraints with or without regular constraints.

One interesting remark is that while our workflow model is closely related to Petri nets, our study of the technical problems uses a myriad of tools including formal languages, automata, linear algebra, and logic.

References

1. Abiteboul, S., Segoufin, L., Vianu, V.: Modeling and verifying active xml artifacts. Data Engineering Bulletin 32(3), 10–15 (2009)
2. Bhattacharya, K., Gerede, C.E., Hull, R., Liu, R., Su, J.: Towards formal analysis of artifact-centric business process models. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 288–304. Springer, Heidelberg (2007)
3. Chao, T., Cohn, D., Flatgard, A., Hahn, S., Linehan, M.H., Nandi, P., Nigam, A., Pinel, F., Vergo, J., Wu, F.Y.: Artifact-based transformation of ibm global financing. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 261–277. Springer, Heidelberg (2009)

4. Chesani, F., Mello, P., Montali, M., Torroni, P.: Verification of choreographies during execution using the reactive event calculus. In: Bruni, R., Wolf, K. (eds.) WS-FM 2008. LNCS, vol. 5387, pp. 55–72. Springer, Heidelberg (2009)
5. NSF Workshop: Research Challenges on Data-Centric Workflows (May 2009), http://dcw2009.cs.ucsb.edu.
6. Deutsch, A., Hull, R., Patrizi, F., Vianu, V.: Automatic verification of data-centric business processes. In: Proc. Int. Conf. on Database Theory (ICDT), pp. 252–267 (2009)
7. Emerson, E.A.: Temporal and modal logic. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B ch. 7, pp. 995–1072. North Holland, Amsterdam (1990)
8. Fritz, C., Hull, R., Su, J.: Automatic construction of simple artifact-based business processes. In: Proc. Int. Conf. on Database Theory, ICDT (2009)
9. Gerede, C.E., Bhattacharya, K., Su, J.: Static analysis of business artifact-centric operational models. In: IEEE International Conference on Service-Oriented Computing and Applications (2007)
10. Garey, M., Johnson, D.: Computers and Intractability A Guide to the theory of NP-Completeness. Freeman, New York (1979)
11. Ginsburg, S.: The Mathematical Theory of Context Free Languages. McGraw-Hill, New York (1966)
12. Ginsburg, S., Spanier, E.: Bounded Algol-like languages. Transactions of AMS 113, 333–368 (1964)
13. Gerede, C.E., Su, J.: Specification and verification of artifact behaviors in business process models. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 181–192. Springer, Heidelberg (2007)
14. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)
15. Ibarra, O.: Private communications (2010)
16. Ibarra, O.H.: Reversal-bounded multicounter machines and their decision problems. Journal of the ACM 25, 116–133 (1978)
17. Küster, J.M., Ryndina, K.: Improving inconsistency resolution with side-effect evaluation and costs. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735, pp. 136–150. Springer, Heidelberg (2007)
18. Küster, J., Ryndina, K., Gall, H.: Generation of BPM for object life cycle compliance. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 165–181. Springer, Heidelberg (2007)
19. Liu, G., Liu, X., Qin, H., Su, J., Yan, Z., Zhang, L.: Automated realization of business workflow specification. In: Proc. Int. Workshop on SOA, Globalization, People, and Work (2009)
20. Lü, Y., Qin, H., Li, J., Chen, Y., Zhang, L., Su, J.: Design and implementation of artimt: An artifact-centric Workflow Management System (2010) (manuscript)
21. Montali, M., Chesani, F., Mello, P., Storari, S.: Towards a decserflow declarative semantics based on computational logic. Technical report, LIA Technical Report 79 (2007)
22. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4) (1989)
23. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal 42(3), 428–445 (2003)
24. Parikh, R.: On context-free languages. Journal of the ACM 13, 570–581 (1966)
25. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice-Hall Inc., Englewood Cliffs (1981)
26. Petri, C.A.: Fundamentals of a theory of asynchronous information flow. In: Proc. of IFIP Congress 62, pp. 386–390. North Holland Publ. Comp., Amsterdam (1963)

27. Presburger, M.: über die Vollständigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchem die addition als einzige operation hervortritt. In: Comptes rendus du premier Congrès des Mathématiciens des Pays Slaves, pp. 92–101, Warszawa (1929)
28. Ryndina, K., Küster, J., Gall, H.: Consistency of business process models and object life cycles. In: Proc. 1st Workshop Quality in Modeling (2006)
29. Singh, M.: Semantical considerations on workflows: An algebra for intertask dependencies. In: Proc. Workshop on Database Programming Languages, DBPL (1995)
30. Singh, M.P., Meredith, G., Tomlinson, C., Attie, P.C.: An event algebra for specifying and scheduling workflows. In: Proc. 4th Int. Conf. on Database Systems for Advanced Applications, DASFAA (1995)
31. van der Aalst, W.M.P.: Business process management demystified: A tutorial on models, systems and standards for workflow management. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets. LNCS, vol. 3098, pp. 21–58. Springer, Heidelberg (2004)
32. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a truly declarative service flow language. In: Proceedings of Workshop on Web Services and Formal Methods (2006)
33. van der Aalst, W.M.P., Pesic, M.: Specifying and monitoring service flows: Making web services process-aware. In: Test and Analysis of Web Services, pp. 11–55. Springer, Heidelberg (2007)
34. Wahler, K., Küster, J.M.: Predicting coupling of object-centric business process implementations. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 148–163. Springer, Heidelberg (2008)