A Monitor Approach in Data-centric BPM

Danfeng Zhao, Guohua Liu, Haibin Liu

Abstract

Business process management (BPM) is a holistic management approach focused on aligning all aspects of an organization. Data-centric BPM is to integrate data and process by combing master data management and business process management. Artifact-centric BPM is a typical kind of data-centered BPM composed by web services. Monitoring is the activity of checking whether at run-time a web service operates according to plan. In this paper, basic definitions in artifact-centric BPM are introduced first. Then, a special FSA is created to represent the semantics of process constraints. Finally, a monitor approach for such data-centric BPM is present which take the special FSA as checking standard and uses a special Snap service to intercept the track messages.

Keywords: BPM; Artifact; Web Service; FSA

1. Introduction

Data-centric BPM is meant to integrate the control aspect with the underlying data. Artifact-centered Business Process Management is a kind of data-centered BPM introduced by IBM [1, 2]. An artifact is a concrete, identifiable, self-describing chunk of information that can be used by a business person to actually run a business [3]. A business artifact consists of an enterprise-wide unique identity and self-describing content. The identity of a business artifact cannot be changed. The content of a business artifact can be modified arbitrarily. Business artifact has lifecycle. A whole lifecycle embodies the implementation of a business process. The definition of Artiflow is introduced by [4], which is an artifact-centered business process logical model. Artiflow describes business process through the attributes of graphs and their interrelations. How to convert Artiflow into physical workflow automatically is also shown here. Paper [5] designs and develops a modeling tool for this Artiflow model.

Web services are described in a standard definition language and communicate with each other requesting execution of their operations in order to collectively support a common business process [6-8]. BPEL (Business Process Execution Language) is the most widely used solution for cooperation amongst web services [9]. We cannot assume that the cooperation between the involved parties will always work as designed, even if bindings are set when planned and not changed at runtime, the communication infrastructure may be unreliable and services can always be changed by their publishers. So, a run-time service monitoring and validation utility is needed to learn service execution traces. Many such systems have been developed, such as approaches described in [10] use instrumentation to check that web services correctly implement their interfaces. These approaches require access to the source code of web services. Approaches in [11, 12] are monitor correctness of conversations between partners. However, they monitor different kinds of properties.

In this paper, we first introduce some basic formalized definitions in section 2. In section 3, the resource of constraint data of a designed business process is introduced and a special FSA is created based on the data. Section 4 takes artiprocess as elementary unit during monitoring, and further propose a monitoring approach which takes the special FSA as checking standard.

2. Basic definitions and descriptions

(1) $T_p$: a finite set of primitive types, such as int, string...
(2) $X$: a finite set of artifact classes, such as "disembarkation card", "monthly report"...
(3) $A$: a finite set of artifacts attributes, such as "name", "sex"...
(4) $ID_C$: a finite set of artifact identifiers for each $C \in X$
(5) $T$: a finite set of artifact types. $T=T_o \cup X$

(6) $\text{DOM}(t)$: domain of each type $t$ in $T$. (1) if $t \in T_o$ is a primitive type, the domain $\text{DOM}(t)$ is some known set of values (integers, strings, etc.); (2) if $t \in X$ is an artifact type, $\text{DOM}(t) = Id(t)$.

Definition1. An (artifact) class is a 6-tuple $(C, A, \tau, Q, s, F)$ where $C \subseteq X$ is a class name, $A \subseteq A$ is a finite set of attributes, $\tau: A \rightarrow T$ is a total mapping, $Q$ is a finite set of states, and $s \in Q, F \subseteq Q$ are initial, final states (resp.).

Definition2. An artifact (object) of class $(C, A, \tau, Q, s, F)$ is a triple $(o, \mu, q)$ where $o \in \text{IDC}$ is an identifier, $\mu$ is a partial mapping that assigns each attribute $A$ in $A$ an element in its domain $\text{DOM}(\mu(A))$, and $q \in Q$ is the current state. An artifact object $(o, \mu, q)$ is initial if $q = s$ (initial state) and $\mu$ is undefined for every attribute, and final if $q \in F$.

Definition3. A state of artifact class $C$ is an element of set $D_1 \times D_2 \times \ldots \times D_n$, where $A=\{A_1, A_2, \ldots, A_n\}$ is the finite set of attributes of $C$. $\text{Dom}(A)$ is the domain of attribute $A_i$, notes $D_i$. The set of states of $C$ is $D_1 \times D_2 \times \ldots \times D_n$, notes $D^C$. If class $C$ is under nonrestraint, the number of states is $\prod |D_i|$ (the restraints here is the ones about attribute definition, for example a constraint may require several attributes be assigned synchronously.)

So, the boundary alignment of $Q$ in definition1 can be rewritten as $Q \subseteq D^C$.

Definition4. A state class of artifact class $C$ is a set of states described as expression $\{A_o, \text{IsDefine}(A_o)\}$, where $\text{IsDefine}(A_o)$ is a mapping from attributes set $A$ to $\{0, 1\}$, $(A_i \in A)$, used to judge the attributes definition situation, if the attribute $A_i$ has a value, return 1; no value or non-defined, return 0.

For example, an artifact class has two attributes “name” and ”sex”, where $D_{\text{name}}=\{\text{Bill}, \text{Marry}\}$; $D_{\text{sex}}=\{\text{male}, \text{female}\}$. Suppose a state class named “full” notes both the two attributes have values, so $\{\langle \text{name, 1} \rangle, \langle \text{sex, 1} \rangle\}$ describes such a set of states $\{\langle \text{Bill, female} \rangle, \langle \text{Bill, male} \rangle, \langle \text{Marry, female} \rangle, \langle \text{Marry, female} \rangle\}$.

So, each artifact class has more than one state class, each state class corresponding to several concrete states. It is useless for web services to distinguish the values of the input or output artifact attributes. (Except some services have especial requirements.). So, we consider artifacts’ conversion among web services as state classes’ transitions, we call state class as “state_r” for short in the following.

Definition5. A service is a 5-tuple $(n, V_r, V_o, P_o, P_n)$, where $n \in S$ is a service name, $V_r$ is a finite set of artifact classes being read only, $V_o$ is a finite set of artifact classes being rewritten, $P_o$ is an expression to the states of input artifacts, $P_n$ is an expression to the state_cs of output artifacts. $P$ can be noted as $\{A_o, \text{IsDefine}(A_o)\}$.

Definition6. The lifecycle of artifact used to describe the end-to-end processing of a specific artifact from creation to completion and archiving.

Artifact lifecycle is a static concept which uses a set of state_cs to describe the sequential steps to archive the process function based on the actual business process rules.

Definition7. An artiprocess expresses an implementation course of the corresponding artifact lifecycle on a certain web services set.

Artiprocess is a dynamic concept which is created dynamic, scheduled to execute and finally die out. This concept emphasizes implementation process.

The description content of artiprocess includes three aspects:

1. Artifact lifecycle: it is the target steps which artiprocess need to archive.

2. Dataset: it contains all the data an artiprocess needed during implementation. Each artiprocess corresponds to a unique dataset, while each lifecycle corresponds to no less than one artiprocess.

3. Control block: it shows the dynamic features of artiprocess. Because of concurrency of artiprocesses, different artiprocesses share resources and further condition each other. To express the conditionality, artiprocess control block should be created first to manage and control artiprocess when an artiprocess is set up. After accomplishing its mission, artiprocess died out through the release of control block by system.

To obtain such information, we utilize a special service which is inserted between any two web services in business process to intercept process snaps.
Example1. Figure1(a) is a part of Artiflow for purchasing business process designed by Atiflow designer[5]. This segment expresses the implementation of artifact class “material requisition”. Figure1(b) is the simple Serflow transferred from figure1(a), in which the last service is a DAS. In this process segment, artifact class “material requisition” be created at the first web service, then it is transferred to the next one which is always executing repeated (so, the temporary repository before second web service is needless during implementation and it can be taken off in the Serflow according to [15]), and finally this artifact class be stored (in serflow this necessary storage step can be replaced by DAS).

a. Artiflow of a process segment

b. Transferred Serflow

Figure 1. The Artiflow and Serflow descriptions for a simple example

So, this artifact class can be described as $C_{mr}=(C,A,\tau,Q,s,F)$, where

$C$="material requisition";

$A=$\{name, number, price, proposer, censor, conclusion \};

$\tau$: $\tau$(name)=char, $\tau$(number)=int, $\tau$(price)=int, $\tau$(proposer)=char, $\tau$(censor)=char, $\tau$(conclusion) =char, $\tau$(store id)=int;

$Q$=\{empty($s$), filled, examined, stored($F$)\}.

Artifacts (objects) of this artifact class $C_{mr}$ can be described as follow:

$o_1$: $(o_1,\mu_1, q_1)$. $o_1$=001; $\mu_1$: name="table", number ="5", price ="50", proposer="Marry", censor=NULL, conclusion=NULL, store id=NULL; $q_1$="filled".

$o_2$: $(o_2,\mu_2, q_2)$. $o_2$=002; $\mu_2$: name="computer", number ="6", price ="2000", proposer="Lucy", censor="David", conclusion=approved, store id="002347"; $q_2$="stored".

The four state cs: ①"empty":$s$=\{<name,0>, <number,0>, <price,0>, <proposer,0>, <censor, 0>, <conclusion,0>,<store id,0> \}; ② "filled":\{<name,1>, <number,1>, <price,1>, <proposer,1>, <censor,0>,<conclusion,0> ,<store id,0> \}; ③ "examined":\{<name,1>, <number,1>, <price,1>, <proposer,1>, <censor,1>,<conclusion,1> ,<store id,0> \}; ④"stored": $F$=\{<name,1>, <number,1>, <price,1>,<proposer,1>, <censor,1>,<conclusion,1> ,<store id,1> \}

One of the services can be defined as $S_1$: $(n_1, V_{st}, V_{sr}, P_{st}, P_{sr})$, where $n_1$="fill in material requisition"; $V_{st}$=NULL; $V_{sr}$={material requisition}; $P_{st}$=\{< name,0>, <number, 0>, <price, 0>, <proposer,0>, <censor,0>,<conclusion,0>,<store id,0> \} $P_{sr}$={< name,1>, <number, 1>, <price, 1>, <proposer,1>, <censor,0>,<conclusion,0> ,<store id,0> }\}

3. Process constraint and FSA

Each special FSA can be used to represent a set of transition rules of a certain artifact class. In this section, we introduce the resource of the process constraints which is used to set up FSA, and then define a special FAS and its semantic description.
3.1 Process constraint

Definition 8. A process constraint is a mapping from a set of artifact class names to a set of services sequences, notes constraint(C) = (service), where service, is a 5-tuple (n, V_r, V_w, P_i, P_o) (described in definition 5).

The constraint information is essential for creating special FSAs, which is gainable through analyzing Artiflow document and mapping table which expresses the one-to-one correspondence between designed services and actual web services. The basic structures of main elements in Artiflow are shown in figure 2. Figure 2(a) shows the detail information about services includes service name, basic operations and so on; figure 2(b) tells the name and related artifact class information; figure 2(c) presents the sequence of all the services and repository grouped by artifact classes. It is noteworthy that the repositories should be transferred to a special kind of web services which is only used to operate database, such as DAS[15]. So, it is feasible to get the established sequence of web services and the corresponding artifact state_cs to create FSA.

![Figure 2. Basic elements structures in Artiflow](image)

3.2 Formal definition of special FSA

Definition 9. Special FSA is a 6-tuple \( M(Q, \Sigma, \delta, q_0, F) \), where \( Q \) is a finite set of state_cs, for \( \forall q \in Q, q \) is a state of \( M \); \( \Sigma \) is a finite set of service names; \( \delta \) is a transfer function from \( Q \times \Sigma \) to \( Q \). For \( \forall (q, \alpha) \in Q \times \Sigma \), \( \delta(q, \alpha) \) means an input of service \( \alpha \) could change \( M \) from state_c \( q \) to \( t \); \( q_0 \) is a start state_c, \( q_0 \in Q \); \( F \) is finite set of final state_cs or acceptable state_cs, \( F \subseteq Q \), for \( \forall q \in F \), \( q \) is a final state_c of artifact.

According to above definition, formula as following describes the implementation situation of a certain artiprocess, \( \forall q_0 \in Q, \alpha_0 \in \Sigma, \delta(q_0, \alpha_0) = q_j \), it expresses that if the state_c of the related artifact object is \( q_0 \), it will transfer to \( q_j \) if this artiprocess execute in service later.
3.3 Semantic description of special FSA

In general, each artifact class in a certain business process corresponds to a FSA representation where arc labels are web service sets and circles are state cs. Each artifact artiprocess in such an artifact class can be monitored and validated by this corresponding FSA[13].

In these figures, a double-circle denotes a final state of the FSA, a single-circle denotes other state, s1 and s2 denote two distinctive web services, and H is a set of web services which are not explicitly mentioned, m≥1.

![Figure 3](image)

Figure 3. Representation of special FSA

The example above could be described by FSA as $M:=Q, \delta, q_0, F >$ where,

$Q:=\{ \text{empty}(q_0), \text{filled}, \text{examined}, \text{stored}(F) \}$

$\delta:=\{ \text{fill in material requisition}(s_1), \text{examine and approve}(s_2), \text{store material requisition}(s_3) \}$

This graph expresses users could fill in material requisition through $s_1$ more than once until $s_2$ occurs. We can use such a FSA to monitor and validate every artiprocesses of this artifact class to detect abnormal implementation of business process in runtime. The detail method is present in next section.

4. Monitoring and Validation Framework

The framework of detecting a process is shown in figure 4. The monitoring and validating use two kinds of data which come from Process constrains & mapping table database and process records database respectively. The detector utilizes FSA as semantic representation of constraints, uses a special Snap service to intercept the track messages going in and out of services, and utilizes two different methods to check the interaction sequence of web services and the correctness of each web service respectively.
4.1 Basic definitions

$s_d^d$: a designed service
$s_a^d$: an actual web service
$s^d$: a sequence of designed services related to a same artifact class, $s^d=s_1^d$, $s_2^d$, ..., $s_n^d$.
$s_a^d$: a sequence of actual web services which executed by a same artiprocess in turn, $s^d=s_1^a$, $s_2^a$, ..., $s_m^a$.

Definition 10. Mapping function: a mapping from a set of designed services to a set of actual web services described in Service Metadata Base, notes as $s_i^a=\text{mapping}(s_i^d)$, where $1\leq i\leq n$.

4.2 Detection methods and algorithms

Theorem 1: For random artifact in a process instance, the basic monitoring for its state should according to the principles following: (1) If $s^d=s^d$, the artifact has finished correctly; (2) If $s^a\subset s^d$, the artifact is correctly running; (3) others, the artifact is in error.

Proof:

All the operations information of designed services has been defined in Artiflow, and designed and accrual services have been one-to-one mapped by certain function. That is to say, $s^d$ can be get from Artiflow which describes the whole designed business process; $s^a$ is actual data intercepted by the Snap Service, which is track record of an artifact at runtime. The change of attributes in each artifact can be found from process record data which is intercepted by Snap Service.

The running status can be generally classified to: finished correctly, running correctly, running in error and ending at error.

A string $w$ is a prefix of a string $x$, denoted $w\subset x$, if $x=wy$ for some string $y\in\Sigma^*$ denoted $w=x$, if $w\subset x$, $|w|=|x|^{[14]}$. So, if $s^a=s^d$, the two sequences are equal and the artifact has finished correctly; if $s^a\subset s^d$, $s^a$ is a prefix of $s^d$, the artifact is running correctly; others, error.

Algorithm 1: Detecting Business Process 1(DBP1)

Input: artifact class name $x$;
Output: judging result variable $r$.

$$\text{DBP1}(x)$$

Begin
get $s_1^d$, $s_2^d$, ..., $s_n^d$ where artifact class name=$x$;
get $s_1^a$, $s_2^a$, ..., $s_m^a$ where artifact class name=$x$;
$s_1^a=\text{mapping}(s_1^d)$;
if $s^a=s^d$
$r='the artifact $x$ has finished correctly'$$
else if $s^a\subset s^d$
$$
r = 'the artifact x is correctly running';
else
r = 'the artifact x is in error';
Return r;
End

This Algorithm aims to judge the state of a certain artifact. The main part is a comparison function, so it has lower time complexity.

In most cases, actual web services are invoked according to designed sequence correctly. But it is not means the all the web services executing inside correctly. The operations of web services act on the special attributes of artifact, only the change of attributes can judge the executing correctness of web services, so we use algorithm2 to achieve this validation.

Algorithm 2: Detecting Business Process 2 (DBP2)
Input: artifact class name x,
Output: r.
get the artiprocess information of artifact class x through Snap service;
lp = artiprocess name;
judge the state_c of artiprocess lp, notes sc={<A, IsDefine(A)>};
get the sequence of executed web services on lp, notes \(s_1^d, ..., s_l^d, ... s_m^d\);
for( i=1, i<=m, i++)
{
  \(s_i^a=\text{mapping}(s_i^d)\);
  get the information of actual web service \(s_i^a\);
p_o = {<a, IsDefine(a)>} is the state_cs of output x corresponding to lp;
  for( k=1, k<=n, k++)
  {
    find \(a = A_k\);
    If(IsDefine(a) == IsDefine(A_k)>)
    break;
    else
    return r='web service' \(s_i^a\) ‘error!’ ;
  }
}

This Algorithm is given to illustrate the validity of implementation of services on a certain artiprocess. There are two loops in it, the time complexity is O(mn), where m is the number of executed web services on this artiprocess, n is the number of attributes in artifact class. Both m and n are constants.

4.3 Experiments

Our framework has been implemented as a part of a prototype written in Java which is introduced in detail in [16]. We created two business processes using this prototype, one is process of Restaurant; another is Commercial Apartments Preselling Approval (CAPA). Most services used to compose them are custom made for special artifacts requirements. We have applied our framework to these services and report on the results of monitoring.
We did not detect errors in running the prototype system, but in order to exercise the monitoring framework, we manually introduced several errors into the web service implementation. For example, we modified web service \textit{payment} to skip the key handler for dealing with artifact PAF. The monitoring framework was able to detect a violation of the artifact state and report user an error warning. As shown in figure 5, the monitor returns two kinds of results. One denotes the process is running accurately, another shows an error message which includes the mistaken service and a right services sequence.

5. Conclusions

Generally, process constraints are specified by designers and are not expected to be violated. However, whether the actual runtime service interaction conforms to the constraints is another question. So, run-time service monitoring and validation is needed. We present an approach and associated techniques that validate the correctness of web services according to process constraints, and our further research is about how to choose satisfied web services to replace mistaken ones at runtime.

6. Acknowledgment

The work is supported by the National Natural Science Foundation of China (No. 61070032).

7. References

[8] Xiaoyong Mei, Aijun Jiang, Shixian Li, Changqin Huang, Xiaolin Zheng, Yiyan Fan, "Formal Modeling Execution Semantic of Control and Data Flow for the Composition Web Services", A Monitor Approach in Data-centric BPM
Danfeng Zhao, Guohua Liu, Haibin Liu


