IBM Research Report

Towards Formal Analysis of
Artifact-Centric Business Process Models

Kamal Bhattacharya[1], Cagdas Gerede[2], Richard Hull[3], Rong Liu[1], Jianwen Su[2]

[1]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

[2]University of California at Santa Barbara

[3]Bell Labs, Alcatel-Lucent

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Towards Formal Analysis of
Artifact-Centric Business Process Models

Kamal Bhattacharya[1], Cagdas Gerede[2], Richard Hull[3], Rong Liu[1], Jianwen Su[2*]

[1] IBM T.J. Watson Research Center   [3] Bell Labs, Alcatel-Lucent
[2] University of California at Santa Barbara

Abstract. Business process (BP) modeling is a building block for design and manage-
ment of business processes. Two fundamental aspects of BP modeling are: a formal
framework that well integrates both *control flow* and *data*, and a set of tools to assist all
aspects of a BP life cycle. A typical BP life cycle includes at least a design phase where
the key concerns are around "correct" realization of business logic in a resource con-
strained environment, and an operational phase where a main objective is to optimize and
improve the realization during the execution (operation). This paper is an initial attempt
to address both aspects of BP modeling. We view our investigation as a precursor to the
development of a framework and tools that enable automated construction of processes,
along the lines of techniques developed around OWL-S and Semantic Web Services.

Over the last decade, the idea of coupling control and data in an "artifact-centric"
approach emerged in the practice of business process design. The key focus is on the
"moving" data as they are manipulated throughout a process. The idea works well with
many applications, typical examples include customer order processing, insurance claim
handling, etc. Based on this idea, we formulate a theoretical model for artifact-centric
business processes and develop complexity results concerning static analysis of three
problems of immediate practical concerns. The problems focus on the ability to complete
an execution, existence of an execution "deadend", and redundancy. It is shown that the
problems are undecidable in the general case; and under various restrictions they are
decidable but complete in PSPACE, co-NP, and NP; and in some cases decidable in linear
time.

1   Introduction

In recent years, competitive business environment has forced companies to be opera-
tionally innovative in order to outperform their competitors [7]. This challenge requires
business process models not only to ensure work to be done as desired but also to en-
able operational innovations. In general, a process model describes activities conducted
in order to achieve business goals, informational structure of a business, and organiza-
tional resources. Workflows, as a typical process modeling approach, often emphasize
the sequencing of activities (i.e., control flows), but ignore the informational perspec-
tive or treat it only within the context of single activities. Without a complete view of
the informational context, business actors often focus on what should be done instead
of what can be done, hindering operational innovations [7, 1].

The goal of this paper is to develop and investigate a new modeling framework
centered around "business artifacts". Business artifacts (or simply artifacts) are the in-
formation entities that capture process goals and allow for evaluating how thoroughly

---

[*] Contact author, su@cs.ucsb.edu

these goals are achieved. Business "services" (or tasks) act on artifacts and then modify artifacts based on business rules. A fundamental thesis of this paper is that business services are typically less frequently changed; a technical challenge is to properly "chain" the services together or "evolve" the current workflow of the services to adapt for new business requirements. In our framework, "business rules" are used to assemble the services together; they are specified in declarative languages and are easy to modify. A process model thus consists of business artifacts, services, and rules. Based on this framework, we study properties of process models that concern both information perspective and control flows. In particular, we develop complexity results on the following problems: Can a process complete, i.e., an artifact can be successfully processed? Does a dead-end path exist in a process? Are there redundant data in artifacts or redundant services?

This paper makes the following contributions.

1. The development of a formal artifact-based business model and a declarative semantics based on the use of business rules which can be created and modified separately from the artifacts.
2. A preliminary set of technical results concerning statically analysing the semantics of a specified artifact-based business process model. The results range from
   (a) undecidability for the general case,
   (b) PSPACE-complete when no new artifacts are created during the execution,
   (c) intractable (co-NP- or NP-complete) under various restrictions including "monotonic" services, and
   (d) linear time under more limited restrictions.

Organization: §2 provides a motivating example for business artifacts and modeling. A formal artifact-centric process model is introduced in §3. In §4, we present technical results on static analysis of process models specified in our framework. §5 discusses related work. §6 concludes this paper with a brief discussion on future work.

2   A Motivating Example

Consider an IT service provider aimed to provide IT services to an enterprise comprising a large number of geographically distributed "small sites". Provided services include IT provisioning, installation, and maintenance as well as general support. Typical examples for small sites are individual hotels part of a larger chain or fast food restaurants that are part of a franchise. The service provider typically signs a contract with the franchise corporation, which determines the service level agreements for each order of a given IT service. For example, a hotel corporation might sign a contract with a service provider that allows the provider to execute any kind of IT systems related services at individual hotel sites. The service provider receives a request for service from a hotel site and creates an order. Fig. 1 illustrates the high level business process for a particular corporation. Typically, after initiating the accounting (e.g. performing a credit check and filing the invoice) a schedule for executing the requested services will be planned (e.g. performing an update on all cash registers in the hotel and an upgrade of the main reservation terminals). Insufficient funds may lead to a termination of the order (before the schedule is created). After the hotel corporation approves the install

for the hotel site, the IT services can execute the planned schedule and complete the order.
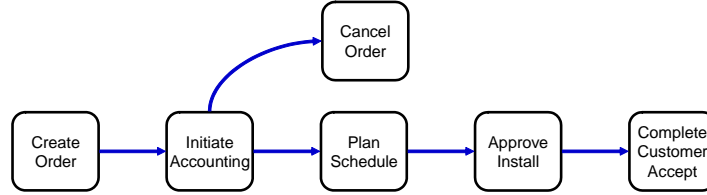

Fig. 1. An IT Service Provider's Process Model

The high-level process model describes the sequence of activities executed by the service provider to reach the business goal of completing the IT services delivery. Each task in the process model describes business intent. In our context it is convenient to consider each task as a service which requires an input, produces an output, may have pre-conditions and have an effect on an external system. (This is similar in spirit to semantic web services, e.g., [4, 12].) In recent years, some of the authors have investigated a different approach to modeling business operations. The artifact-centered approach has been documented in various papers [3, 15, 11, 6], and has been applied in various both internal and external IBM client engagements. The key idea is to shift the focus of business modeling from the actions taken to the entities that are acted upon. These entities (business artifacts or simply artifacts), such as an Order, a Request, a Plan, or an Invoice are information entities used by enterprises to keep track of their business operations. Not every information entity in a business is a business artifact. The focus for artifact-centric modeling is on business entities that (a) are records used to store information pertinent to a given business context, (b) have a distinct life-cycle from creation to completion, and (c) have a unique identifier that allows identification of an artifact across the enterprise. Business artifacts are an abstraction to focus businesses on not just any information but on the core information entities that are most significant from a perspective of accountability. The artifact is an information record that allows for measuring whether or not the business is on track to achieve their business goals. [3, 15, 11] provides more information on the methodology to identify business artifacts.

In the previous example, the key business artifact is the Order. The Order stores different aspects such as the date created, the result of the credit check, planned execution date, etc. The Order exists in different states or stages such as Pending Order, Planning, Live Order and Completed. The services will interact with the artifacts by (a) instantiating an artifact instance, (b) updating an artifact, and (c) triggering state transitions on artifacts. We require that each service will at least update one or many business artifacts, or change the state of at least one artifact. The reasoning behind the update requirement is the intent to model for accountability; hence, a business user should be empowered to measure what has been done in each step of a given process. Figure 2 illustrates the artifact-centric modeling approach using the service delivery example.

The Create Order service creates an instance of the Order artifact, updates the artifact (update dateCreated) and triggers the transition from Pending Order into Planning. The Initiate Accounting service interacts with the Invoice artifact, which captures information pertinent to accounting. The result of this interaction is (typically) a validation of a sufficient credit. Business rules are used to either cancel the order or instantiate the
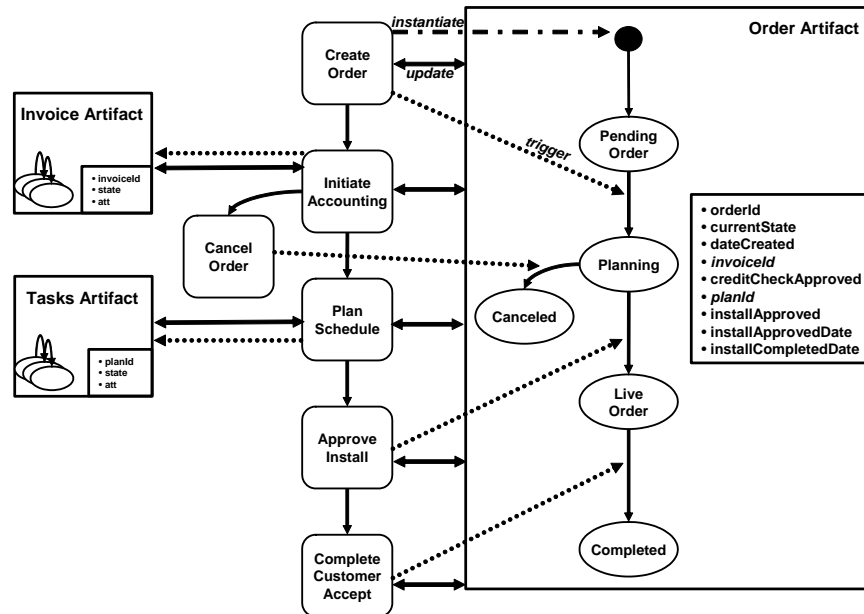
Fig. 2. An artifact-based view of the same workflow

Plan Schedule service. The Initiate Accounting will update the Order artifact with the appropriate decision (update creditCheckApproved with either true or false). The Plan Schedule service will create an execution plan by interaction with the Tasks artifact. Neither the Initiate Accounting service nor the Plan Schedule services trigger a state transition but both update the Order artifact.

The three artifacts shown in Fig. 2 are related to each other. The information content of the Order artifact references both the Invoice artifact and the Tasks artifact. This implies traceability of Tasks and Invoices from an Order artifact. There are different models for artifact relationships. In the context of this paper we omit details about the different models, but assume for simplicity that each artifact in the given business context is associated to other artifacts in the same business context.

The abstraction of artifacts and services that act upon artifacts facilitate customization of services flows. In an engagement scenario, identifying artifacts of a business precedes the definition of the services flows that will execute the operations on artifacts. The sequence of services can be exchanged unless some constraints prevent it, e.g., because of a data dependency between services. To illustrate customization, suppose that for Hotel Corporation A the contract between the service provider and A requires to perform a credit check before the plan execution as the contract requires each hotel site to provide their own budget for maintenance services. For Hotel Corporation B, which centrally budgets maintenance, planning the execution schedule is the only action required, hence, Initiate Accounting can be skipped altogether as accounting is covered in the overall contract. Finally, for Hotel Corporation C may be charged by the service provider on a per-site and per-service level. Thus, Corporation C expects a quote before each transaction, and the schedule is planned prior to accounting. We believe that the challenges one typically encounters for building systems that allow for on-demand cus-

tomization can be overcome with a shift of focus from a activity (or verb)-centric to an artifact (or noun)-centric perspective.

Recent articles [3, 15, 11] on artifact-centric modeling have described various techniques and experiences to support business transformation through the design of artifact-based operation models, and some [15, 11] have laid out how to graphically represent business operations using artifacts. The artifact-centric approach has been applied in various client engagements at IBM and has been used both for business analysis and business-driven development engagements. Our goal in this paper is to define a more generic formal model that provides us with the tools to reason about some very practical problems encountered in real world engagements.

In order to help lay the groundwork for the formal study of the artifact-based approach to business processes, we study three fundamental issues in this paper.

Issue 1: As with any other business process modeling technique, artifact-centric operation models can get quite complex, especially when a large number of artifacts are involved to describe a business scenario. In this case it is helpful to perform a reachability analysis to support the needs of both business and solution architects. A reachability analysis will allow to determine if an artifact is processed properly through its life-cylce from creation to completion.

Issue 2: An artifact can reach a valid state, which is a dead-end, i.e. a final state which is not the completed state. Most of the times, these dead-end paths exist by design, as e.g. in Figure 2 where the canceled state is introduced by design. Techniques to detect non-intended dead-end situations are valuable especially if one can provide guidance how to resolve the situation.

Issue 3: Any process modeling engagement is iterative. The artifact-based approach can be a valuable tool in business transformation, because it allows a business to make the key data in their processes visible. In practice, however, there is a tendency to keep all of the data used by the legacy processes in the artificats as they are designed. This however may lead to the design of services that act upon on potentially redundant data. In this case it is useful to have tools that reason about the pre-design of the business artifact, to reduce redundancy and to re-organize the services flows to remove unnecessary attributes and tasks/services.

3 Formal Model for Artifact-centric Business Processes

In this section we introduce the formal model for business processes. The key notions include, artifacts, schemas, services, and business rules. These are brought together by the notion of "artifact system".

3.1: Artifacts and schemas. To begin with, we assume the existence of the following pairwise disjoint countably infinite sets: $\mathcal{T}_p$ of *primitive types*, $\mathcal{C}$ of *(artifact) classes (names)*, $\mathcal{A}$ of *attributes (names)*, STATES of artifact *states*, and ID_C of *(artifact) identifiers* for each class $C \in \mathcal{C}$. A *type* is an element in the union $\mathcal{T} = \mathcal{T}_p \cup \mathcal{C}$.

The *domain* of each type t in $\mathcal{T}$, denoted as DOM(t), is defined as follows: (1) if $t \in \mathcal{T}_p$ is a primitive type, the domain DOM(t) is some known set of *values* (integers, strings, etc.); (2) if $t \in \mathcal{C}$ is an artifact type, DOM$(t) = \text{ID}_t$.

5

Definition. An *(artifact) class* is a tuple $(C, \boldsymbol{A}, \tau, Q, s, F)$ where $C \in \mathcal{C}$ is a class name, $\boldsymbol{A} \subseteq \mathcal{A}$ is a finite set of attributes, $\tau : \boldsymbol{A} \to \mathcal{T}$ is a total mapping, $Q \subseteq \text{STATES}$ is a finite set of states, and $s \in Q, F \subseteq Q$ are *initial*, *final* states (resp.).

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

ARTIFACT CLASS ORDER		AN ORDER OBJECT
STATES:	ATTRIBUTES:	ID: *id927461*
`PendingOrder (initial)`	`invoice: Invoice`	STATE: *LiveOrder*
`Planning`	`task: Task`	ATTRIBUTES:
`Canceled`	`dateCreated : String`	`invoice:` *id1317231*
`LiveOrder`	`crediCheckApproved : bool`	`task:` *id540343*
`Completed (final)`	`currentCredit: int`	`dateCreated:` *"2 April 2007"*
`...`	`installAprroved: bool`	`creditCheckApproved:` *true*
	`...`	`currentCredit:` *undefined*
		`installApproved:` *undefined*
		`...`

Fig. 3. Order Artifact and an Order Object

Example 3.1 Fig. 3 illustrates the Order artifact class from the example of Section 2, along with an object of this class. As defined by that class, an order has a record of important business information such as the date the order is created, and has references to the related artifacts such as Invoice and Task. The class description also contains possible states an order can be in such as "Pending Order". An artifact of the Order class has three parts: "ID" maps to a unique identifier for the artifact; "STATE" maps to one of the states defined by the Order class; and each attribute maps to either *undefined* or a value in the domain of the attribute's type. ∎

When the context is clear, we may denote an artifact class $(C, \boldsymbol{A}, \tau, Q, s, F)$ simply as $(C, \boldsymbol{A})$, or even C. A class C_2 is *referenced by* another class C_1 if an attribute of C_1 has type C_2. Similarly, an identifier o is *referenced in* an artifact $\boldsymbol{o}$ if o occurs as an attribute value of $\boldsymbol{o}$.

An artifact $\boldsymbol{o}'$ *extends* another artifact $\boldsymbol{o}$ if (1) they have the same identifier and (2) the partial mapping of $\boldsymbol{o}'$ extends that of $\boldsymbol{o}'$ (i.e. if an attribute is defined in $\boldsymbol{o}$, then it is also defined in $\boldsymbol{o}'$ and has the same value as in $\boldsymbol{o}$).

Note that an artifact may have some of its attributes undefined.

Definition. A *schema* is a finite set Γ of artifact classes with distinct names such that every class referenced in Γ also occurs in Γ.

For technical convenience, we may assume that classes in a schema have pairwise disjoint sets of states. Note that distinct classes have disjoint sets of identifiers.

Let S be a set of artifacts of classes in a schema Γ. S is *valid* if artifacts in S have distinct identifiers; S is *complete* if every identifier of type C referenced in an artifact in S is an identifier of another artifact in S.

Definition. Let Γ be a schema. An *instance* of Γ is a mapping I that assigns each class C in Γ a finite, valid, and complete set of artifacts of class C. Let $inst(\Gamma)$ denote the

set of all instances of Γ. An instance $I \in inst(\Gamma)$ is *initial*, *final* (resp.) if every artifact in I is initial, final (resp.).

Notation. Let J be a collection of artifacts and o an identifier of an artifact in J. We denote by $J(o)$ the artifact $\boldsymbol{o} = (o, \mu, s)$ in J; and by $J(o).A$ or $\boldsymbol{o}.A$ the value $\mu(A)$, where A is an attribute of $\boldsymbol{o}$.

Example 3.2 Consider a schema consists of 3 artifacts: Order, Task, and Invoice, as in Section 2. An instance I of this schema is $I(Order) = \{\boldsymbol{o_1}, \boldsymbol{o_2}\}$, $I(Task) = \{\boldsymbol{o_3}\}$, $I(Invoice) = \{\boldsymbol{o_4}\}$, where $\boldsymbol{o_1}$ is an order artifact illustrated in Fig. 3 such that "$\boldsymbol{o_1}$.task" holds the ID of $\boldsymbol{o_3}$, and "$\boldsymbol{o_1}$.invoice" holds the ID of $\boldsymbol{o_4}$; whereas, "$\boldsymbol{o_2}$.task" and "$\boldsymbol{o_2}$.invoice" are undefined. ∎

3.2: Services and their semantics. We now proceed to modeling "services". These are essentially existing software modules used to act on artifacts, and serve as the components from which business models are assembled. We assume the existence of pairwise disjoint countably infinite sets of variables for classes in $\mathcal{C}$. A variable of type $C \in \mathcal{C}$ may hold an identifier in ID_C.

Definition. The set of *(typed) terms over* a schema Γ includes the following.
- Variables of a class C in Γ, and
- $x.A$, where x is a term of some class C (in Γ) and A an attribute in C. (Note that $x.A$ has the same type as A.)

Roughly, a "service" is described by input variables, a precondition, and conditional effect. This can be viewed as a variation on the spirit of OWL-S [4, 12], where services have input, output, precondition, and conditional effects. In the case of OWL-S, the precondition and effects may refer to the inputs and outputs, and also to some underlying "real world". In our context, the artifacts correspond to the OWL-S "real world", and the conditional effects typically write new values into the artifacts (or change their state) – it is for this reason that we don't explicitly specify the outputs of services. The preconditions and conditional effects are defined using logical formula characterizing properties of the artifacts before and after the service execution. In the present paper, we focus on the static analysis of artifacts and their associated processing flow, and thus do not model repositories explicitly. Reference [6] presents a model that models and studies the artifact repositories in a much more refined manner.

We now define the notions of "atoms" and "conditions", which are used to specify the preconditions and conditional effects..

Definition. An *atom over* a schema Γ is one of the following:
1. $t_1 = t_2$, where t_1, t_2 are terms of class C in Γ,
2. DEFINED(t, A), where t is a term of class C and A an attribute in C,
3. NEW(t, A), where t is a term of class C and A an artifact typed attribute in C, and
4. $s(t)$ (a *state* atom), where t is a term of class C and s a state of C.

A *negated atom* is of form "$\neg c$" where c is an atom. A *condition over* Γ is a conjunction of atoms and negated atoms. A condition is *stateless* if it contains no state atoms.

Let I be an instance of a schema Γ. An *assignment (resp. for I)* is a mapping from variables to IDs (resp. occurring in I) such that a variable of class C is mapped to an ID in ID_C. Under a given instance I and a given assignment ν for I, all variables

are assigned (identifiers of) artifacts, the *truth value* of a condition is defined naturally (details omitted) with the following exceptions: (1) DEFINED(t, A) is true if the attribute A of the artifact t has a value, and (2) NEW(t, A) is true if the attribute A of the artifact t holds an identifier not in I. If ν is an assignment for an instance I, we denote by $I \models \varphi[\nu]$ if under the assignment ν, the artifacts in I satisfies a condition φ. Let $\mathbb{V}$ denote the set of all assignments.

Following the spirit of OWL-S, we provide a mechanism to specify the "effect" of executing a service. In our case, the effect will be described in terms of how it impacts whether artifact attributes become defined or undefined, and whether new artifacts are created. We follow the spirit of OWL-S in allowing non-determinism in the model – execution of a service might result in one of several possible effects – this corresponds to the intuition that our model is fairly abstract, and does not encompass many relevant details about the underlying artifacts.

Let V be a set of variables of classes in a schema Γ. A (*conditional*) *effect* over V is a finite set $E = \{\psi_1, \ldots, \psi_q\}$ of stateless conditions over V. In this context we call each ψ_p a *potential effect* of E. Intuitively, if a service s with conditional effect E is applied to an instance I, then the resulting instance will satisfy ψ_p for some $p \in [1..q]$. We also incorporate a condition based on the notion of circumscription [16] to capture the intuition that "nothing is changed in the input instance except things required to satisfy ψ_p". (This contrasts with the approach taken by OWL-S, in which the effect portion of a conditional effect is interpreted using the conventional logic semantics rather than one based on circumscription.)

We can now describe services and their semantics. We assume the existence of a disjoint infinite set $\mathbb{S}$ of *service names*.

Definition. A *service* over a schema Γ is tuple (n, V_r, V_w, P, E), where $n \in \mathbb{S}$ is a service name, V_r, V_w finite sets of variables of classes in Γ, P a stateless condition over V that does not contain NEW, and E a conditional effect.

Intuitively, V_r, V_w are artifacts to be read, modified (resp.) by a service. (These may overlap). Note, however, that if $v \in V_r$, then terms such as $v.A.B$ can be used, for some artifact-valued attribute A, to read attribute values associated with artifacts lying outside of the image of V_r under an assignment ν. The analogous observation holds for V_w. It is possible to prevent this through syntactic restrictions.

```
SERVICE UPDATECREDIT                    SERVICE PLANSCHEDULE
WRITE:{x:Order}                         WRITE:{x:Order}
READ:{y:CreditReport}                   READ:{x:Order, s:Supplier, c:Site}
PRE: ¬ DEFINED(x, creditCheckApproved)  PRE: ¬ DEFINED(x.task)
    and ¬ DEFINED(x, currentCredit)     EFFECTS:
EFFECTS:                                  - NEW(x, task) and
  - DEFINED(x, creditCheckApproved)        DEFINED(x.task, expectedStartDate) and
  - DEFINED(x, creditCheckApproved) and    DEFINED(x.task, expectedEndDate) and
  DEFINED(x, currentCredit)                DEFINED(x.task, supplier) and
                                           DEFINED(x.task, site) and
                                           x.task.supplier = s and x.task.site = c
```

Fig. 4. Example Services

Example 3.3 Fig. 4 illustrates two services. Service UpdateCredit updates an order's credit information according to the credit report. In some cases, when the credit check is approved, the credit amount is not known at the time of the update; in those cases, only

the credit check approved field is defined. This is modeled with two possible effects. Service PlanSchedule creates a task for an order and defines attributes of the task such as expectedStartDate and supplier. ∎

Definition. Let $\sigma = (n, V_r, V_w, P, E)$ be a service over a schema Γ. The (*circumscribed*) *semantics* of σ is a set $[\![\sigma]\!] \subseteq \mathbb{V} \times inst(\Gamma) \times inst(\Gamma)$ such that for each $I \in inst(\Gamma)$ and assignment ν for I over $V_r \cup V_w$,

(1) There is at least one J with $(\nu, I, J) \in [\![\sigma]\!]$ iff $I \models P[\nu]$ (i.e., I satisfies the pre-condition P under assignment ν), and

(2) if $(\nu, I, J) \in [\![\sigma]\!]$ then there is some potential effect $\psi \in E$ for which there exist sets K_{prev} and K_{new} of artifacts over schema Γ having disjoint sets of distinct artifact IDs such that:

   (a) $K = K_{prev} \cup K_{new}$ is an instance of Γ.
   (b) The collections of artifacts in I and in K_{prev} have the same set of identifiers. (This implies that ν is an assignment for K.)
   (c) For each artifact ID o occuring in I, the artifact class and state of o in K_{prev} is identical to the artifact class and state of o in I.
   (d) For each atom in ψ of form $\text{NEW}(t, A)$ there is a distinct artifact ID o in K_{new}, such that $\nu(t.A) = o$. Further, each artifact ID o in K_{new} corresponds to some atom of form $\text{NEW}(t, A)$ occurring in ψ.
   (e) For each artifact $o \in \text{ID}_C$ in K_{new}, o is in the start state for C.
   (f) ("Satisfaction") $(K \cup L) \models \psi[\nu]$.
   (g) ("Circumscription") Suppose that $o \in \text{ID}_C$ occurs in K, and let A be an attribute of C. Suppose that $o \neq t[\nu]$ for any term t which occurs in an atom of ψ that has any of the following forms:
      (i) $t.A = t'.B$ or $t'.B = t.A$ for some attribute B;
      (ii) $\text{DEFINED}(t, A)$ occuring in ψ;
      (iii) $\text{NEW}(t, A)$
      Then we have the following
      (α) If o occurs in I, then $o.A$ is defined in K iff $o.A$ is defined in I.
      (β) If o occurs in K_{new}, then $o.A$ is undefined in K.

Intuitively, the set K_{prev} in the above definition captures the way that existing artifacts in I are changed by potential effect ψ, and the set K_{new} corresponds to new artifacts that are created by ψ. The circumscription condition ensures that if $(\nu, I, J) \in [\![\sigma]\!]$, then an attribute value is changed in J only if this is required in order to satisfy ψ.

Concrete business services will assign specific values for attributes, or might invalidate an existing value, with the result of making it undefined again. In our abstract model, we focus only on whether the service gives a defined value to an attribute, or makes the attribute undefined again. It is useful to consider services that are "monotonic", by which is meant that each attribute can be written at most once (and not reassigned nor moved back to the undefined condition). Artifact schemas with monotonic services enjoy certain decidability and complexity properties. Further, in many real situations the underlying business artifacts are in fact monotonic, due to the need for historical logging. (Typically, some attributes of the artifact schemas in those situations are set- or list-valued, so that multiple "draft" values for an attribute can be assigned

before a final value is committed to. An analysis of the impact of including such value types in the model is beyond the scope of the current paper.)

Definition. Let I and J be instances of an artifact schema Γ. Then J *extends* I if for each artifact ID o in I, o occurs in J and $J(o)$ extends $I(o)$.

Definition. A service σ is *monotonic* if J extends I for each $(\nu, I, J) \in [\![\sigma]\!]$.

For the technical development, it is useful to work with services which affect just one attribute value.

Definition. The service $\sigma = (n, V_r, V_w, P, E)$ is *atomic* if it is monotonic, $V_w = \{x\}$ is a singleton set, and for each $(\nu, I, J) \in [\![\sigma]\!]$, I and J differ only in the following ways:
(a) For at most one artifact ID o and one attribute A of o, $I(o).A$ is undefined and $J(o).A$ is defined.
(b) If in (a) the type of A is artifact class C, then J has one artifact ID that I does not, namly $J(o).A$.
(c) The set of artifact IDs in I is contained in the set of artifact IDs in J.

The service σ is *scalar atomic* if the attribute changed is of a primitive type in $\mathcal{T}_p$.

3.3: Business rules. Based on an artifact schema and a set of available services, a business model is then formulated by "business rules", which define the business logic. Roughly, business rules can specify what services are to be excuted on which artifacts and when.

Technically, we assume some fixed enumeration of all variables. If σ is a service, we will use the notation $\sigma(x_1, ..., x_\ell; y_1, ..., y_k)$ to mean that $x_1, ..., x_\ell$ is an enumeration of variables in the modify set and $y_1, ..., y_k$ an enumeration of read only variables (i.e., in the read set but not the modify set).

Definition. Given a schema Γ and a set of services $\boldsymbol{S}$, a *business rule* is an expression with one of the following two forms:
– "**if** φ **invoke** $\sigma(x_1, ..., x_\ell; y_1, ..., y_k)$", or
– "**if** φ **change state to** ψ".

where φ is a condition over variables $x_1, ..., x_\ell, y_1, ..., y_k$ ($\ell, k \geqslant 0$), σ a service in $\boldsymbol{S}$ such that $x_1, ..., x_\ell$ are all artifact variables to be modified and $y_1, ..., y_k$ are all read only variables of σ, and ψ a condition consisting of only positive state atoms over $x_1, ..., x_\ell$.

```
If PendingOrder(x) and DEFINED(x, creditCheckApproved)
    invoke InitiateAccounting(x;)
If DEFINED(x.task.expectedStartDate) and
   DEFINED(x.task.expectedEndDate) and
   DEFINED(x, installApproved)
   change state to LiveOrder(x) and PendingTask(x.task)
```

Fig. 5. Example Business Rules

Example 3.4 Fig. 5 illustrates two business rules. The first rule says if an order is in PendingOrder state and the credit check is approved, then the service InitiateAccounting is invoked. The second rule says if for an order, the expected start date and the expected end date of the associated task are defined, and the installation is approved, then the order moves to LiveOrder state, while the associated task moves to PendingTask state. ∎

We now briefly describe the semantics of business rules. For two given instances I, J of a schema Γ, and a given assignment $\nu \in \mathbb{V}$, a business rule r, we say I *derives* J *using* r *and* ν, denoted as $I \overset{r, \nu}{\rightarrow} J$, if one of the following holds.

- $I \models \varphi[\nu]$ and $(\nu, I, J) \in [\![\sigma]\!]$, if r is the rule "**if** φ **invoke** $\sigma(x_1, ..., x_\ell; y_1, ..., y_k)$".
- $I \models \varphi[\nu]$ and J is identical to I except that each $J(\nu(x_i))$ has the state according to ψ, if r is the rule "**if** φ **change state to** ψ".

3.4: Artifact systems and their semantics. A business model is then captured as an "artifact system".

Definition. An *artifact system* is a triple $W = (\Gamma, \boldsymbol{S}, \boldsymbol{R})$ where Γ is a schema, $\boldsymbol{S}$ is a family of services over Γ, and $\boldsymbol{R}$ is a family of business rules with respect to Γ and $\boldsymbol{S}$.

In the next section, we shall also include, as a fourth component, a set $\boldsymbol{C}$ of constraints; in these cases we shall indicate the class of constraints from which $\boldsymbol{C}$ is drawn.

We now sketch the semantics of artifact systems, in particular, define the notion focused path for an artifact, which will be the basis for much of the technical investigation.

Definition. Let $W = (\Gamma, \boldsymbol{S}, \boldsymbol{R})$ be an artifact system and C a class in Γ. A *path* in W is a finite sequence $\pi = I_0, I_1, ..., I_n$ of instances of Γ. The path is *valid* if
(i) For each $j \in [1..n]$, I_j is the result of applying one business rule r of $\boldsymbol{R}$ to I_{j-1},
   i.e., $I_{j-1} \xrightarrow{r,\nu} I_i$ for some assignment $\nu \in \mathbb{V}$.
For an artifact ID o, the path π is *o-relevant* if
(ii) $n \geqslant 1$,
(iii) o does not occur in I_0, and
(iv) o does occur in I_1.
(Intuitively, this means that artifact ID o is created in the transition from I_0 to I_1.) If the path is o-relevant, then it is *o-successful* if $I_n(o)$ is in a final state. It is *o-dead-end* if o is not in a final state for C and there is no sequence $I_{n+1}, ..., I_m$ such that $I_0, I_1, ..., I_n, I_{n+1}, ..., I_m$ is a valid, o-successful path. Finally, a valid path $I_0, I_1, ..., I_n$ is *o-focused* if it is o-relevant and
(v) o does occur in I_j for each $j \in [1..n]$
(vi) for each $j \in [2..n]$, we have $I_j(o) \neq I_{j-1}(o)$ (i.e., o has changed state or some attribute value of o has changed).
(vii) for each $j \in [2..n]$, and for each $o' \neq o$, we have $I_j(o') = I_{j-1}(o')$ (i.e., o' does not change state and no attribute value of o' has changed).

We close this section by introducing a formalism that characterizes "redundant attributes". For an attribute A of a class C, let $\rho_A(C)$ represent the class identical to C but without A. For a schema Γ, let $\rho_{C.A}(\Gamma)$ be the schema $(\Gamma - \{C\}) \cup \{\rho_A(C)\}$. If $\boldsymbol{S}$ is a set of services, let $\boldsymbol{S}_{C.A}$ be the set $\{\sigma \mid \sigma \in \boldsymbol{S}$ and σ references A of $C\}$. Similarly, for a set $\boldsymbol{R}$ of business rules, let $\boldsymbol{R}_{C.A}$ be the set $\{r \mid r \in \boldsymbol{R}$ and r references $C.A$, or r invokes a service in $\boldsymbol{S}_{C.A}$ $\}$. Let $W = (\Gamma, \boldsymbol{S}, \boldsymbol{R})$ be an artifact system. Define $\rho_{C.A}(W)$ as the artifact system $(\rho_{C.A}(\Gamma), \boldsymbol{S} - \boldsymbol{S}_{C.A}, \boldsymbol{R} - \boldsymbol{R}_{C.A})$. Intuitively, $\rho_{C.A}(W)$ is an artifact system similar to W but with attribute A of class C completely removed. This removal operation $\rho_{A.C}$ is natually extended to instances, and to paths. For the latter, if the result of $\rho_{A.C}$ on a consecutive block of instances in the path yield identical instances, then all but one of the indentical instances are removed in the resulting path.

Definition. For an identifier o of class C, we say an attribute A of a class C is *redundant on* an [o-focused and] o-successful path π in W if $\rho_{C.A}(\pi)$ is an [o-focused and] o-successful path in $\rho_{C.A}(W)$. An attribute A of a class C is *redundant in W* [for C-focused paths] if A is redundant on every [o-focused and] o-successful path in W.

4 Technical Results

We believe that the artifact model can provide the backbone for the automated construction of workflow schemas, or more specifically the automated construction of artifact schemas. The section studies some basic decision problems about schemas, and obtains complexity characterizations for them. The problems studied are chosen based on our interest in testing whether automatically generated schemas satisfy key reachability and minimality properties.

Due to space limitations, we do not include the proofs of the technical results here.

We now provide formal counterparts to the intuitive decision problems introduced in Section 2. Each of these is of fundamental importance when constructing artifact-based workflows (either manually or automatically). Suppose that $W = (\Gamma, \boldsymbol{S}, \boldsymbol{R})$ is an artifact system (possibly with constraints), and C is an artifact class in Γ.

Q1: (*Successful completion for C*.) Is there an ID_C o and a valid, o-successful path in W?

Q2: (*Dead-end path for C*.) Is there a ID_C o and a valid, o-dead-end path in W? Given W with dead-end paths, is there a way to construct an artifact system W' which (a) is "equivalent" to W (according to a definition given below) and (b) has no dead-end paths for C?

Q3: (*Attribute redundancy for C*). Is an attribute A of C redundant in W?

Intuitively, **Q1** is focused on whether class C in W is "satisfiable". The existance of at least one successful completion for C is a minimum test on whether W is well-formed.

Q2 is based on a more refined notion of well-formed. Suppose that W does have a valid, dead-end, o-focused path. This suggests that an execution of the workflow can reach a point in which the artifact o cannot be further extended to completion. In other words, the workflow would need to perform a "roll-back" for this artifact. To avoid this undesirable situation, it might be possible to construct a new artifact system W' from W which supports all of the same successful paths as W, but which has no dead-end paths. (This might be acheived, for example, by adding constraints to W, which for any successful, non-dead-end path $I_0, ..., I_j$, prevent moves into an instance I_{j+1} for which $I_0, ..., I_j, I_{j+1}$ is valid but dead-end. See Theorem 4.5 below.)

Q3 can be used to assist in optimizing an artifact system. If W has a redundant attribute, then this attribute, all business rules and services referring to this attribute can be removed from W which reduces the complexity of the design specification.

We first note that all three questions are undecidable in the context of general artifact systems.

Theorem 4.1 Let $W = (\Gamma, \boldsymbol{S}, \boldsymbol{R})$ be an artifact system. Then each of **Q1**, **Q2**, and **Q3** for class C is undecidable. When W does not contain predicate NEW, **Q1**, **Q2**, and **Q3** are in PSPACE, and furthermore, they are complete in PSPACE for o-focused paths.

We now look at some restricted forms of artifact systems for which the questions are either tractible or NP-complete.

To obtain various decidability results, we focus herafter on artifact systems that are monotonic. Also, to simplify the discussion, we assume that all artifact systems under consideration are atomic.

Our first result yields tractable decidability for **Q1**. For this result, we use a slight variation on the conditions used elsewhere in the paper. Recall that an atom over schema Γ may have the form $s(t)$ where t is an artifact term of some class C and s is a state of C in Γ.

Definition. A *previous-or-current-state* atom has the form $[prev\_curr]s(t)$. Let ν be a variable assignment and $I_0, I_1, ..., I_n$ a $\nu(t)$-relevant path. Then $[prev\_curr]s(t)$ is true under ν for I_n in the context of $I_0, I_1, ..., I_n$ if $I_j(\nu(t))$ is in state s for some $j \in [1..n]$, (i.e., if $\nu(t)$ is in state s in I_n, or was in state s in some preceding instance of the path).

Theorem 4.2 Let $W = (\Gamma, \boldsymbol{S}, \boldsymbol{R})$ be a monotonic artifact system. Assume that
 (i) Each service S in $\boldsymbol{S}$ is *deterministic*, i.e., it has exactly one conditional effect, whose antecedant is "true".
 (ii) The pre-condition for each service is positive (i.e., no negated atoms), has no atoms of the form $s(t)$ for a state s, but may have atoms of the form $[prev\_curr]s(t)$.
 (iii) The antecedant of each rule in $\boldsymbol{R}$ is positive, has no atoms of the form $s(t)$ for a state s, but may have atoms of the form $[prev\_curr]s(t)$.

Let A be an attribute of a class C in Γ, and o a ID_C. Then there are linear-time algorithms to decide the following.
 (a) For an attribute A of C, whether there is an o-focused, o-successful path $I_0, \ldots, I_n$ in W such that $I_n(o).A$ is defined.
 (b) Whether there is an o-focused, o-successful path $I_0, \ldots, I_n$ in W.

Our next result shows that slight relaxation of most of the conditions in the above theorem yields NP-completeness for **Q1**.

Theorem 4.3 Let $W = (\Gamma, \boldsymbol{S}, \boldsymbol{R})$ be a monotonic artifact system. In connection with monotonic, o-successful paths (which have no artifact invention but which are not required to be o-focused), Question **Q1** is NP-complete for the following cases. (Here conditions (i) through (iii) refer to the conditions of Theorem 4.2
 (a) Conditions (ii), (iii) are satisfied by W but condition (i) is not. Furthermore, each service can be applied at most once to a given artifact.
 (b) Conditions (i), (ii), (iii) are satisfied by W, except that negation is permitted in the pre-conditions of services.
 (c) Conditions (i), (ii), (iii) are satisfied by W, except that negation is permitted in the antecedents of business rules.
 (d) Instead of using previous-or-current-state atoms the rule pre-conditions and conndional effect antecedants may use atoms of the form $s(t)$. All other conditions of Theorem 4.2 apply.

These are all NP-hard even in the case of o-focused paths.

While the various decision problems just mentioned are all NP-complete in the worst case, we expect that heuristics can be developed to decide these problems in commonly arising cases.

In an artifact system $W = (\Gamma, \boldsymbol{S}, \boldsymbol{R})$, the business rules $\boldsymbol{R}$ provide the mechanism for a workflow execution to "make forward progress". In some cases it may also be convenient to specify constraints on the execution, which can succinctly prevent certain rules from executing. A simple form of constraint is now introduced.

Definition. Let Γ be an artifact schema and C an artifact class in Γ. The *undefined-att-state-blocking* constraint for a set $\mathbf{A} = A_1, \ldots, A_n$ of attributes for C and state s for C is the expression $\neg\text{DEFINED}A_1 \wedge \ldots \wedge \neg\text{DEFINED}A_n \rightarrow block$ **change state to** s. A short-hand for this is $\text{UNDEFINED}A \rightarrow block \ s$.

We extend the notion of artifact system to include such constraints.

Definition. An artifact system (*with undefined-att-state-blocking constraints*) is a 4-tuple $W = (\Gamma, \boldsymbol{S}, \boldsymbol{R}, \boldsymbol{C})$ where $(\Gamma, \boldsymbol{S}, \boldsymbol{R})$ is an artifact system as defined before and $\boldsymbol{C}$ is a family of undefined-att-state-blocking constraints over Γ. Each path $\pi = I_0, ..., I_n$ for $W' = (\Gamma, \boldsymbol{S}, \boldsymbol{R})$ is also a path for W. This path is *valid* for W if it is valid for W' and for each $j \in [1..n]$ and each constraint $\text{UNDEFINED}A \rightarrow block \ s$ in $\boldsymbol{C}$, if the transition from I_j to I_{j+1} includes moving an artifact o into class s, then $I_j(o).A$ is defined for some $A \in \mathbf{A}$.

As it turns out, a system with blocking constraints can be replaced by an "equivalent" system without constraints. but there there may be an exponential blow-up in the size of the system.

Definition. Let $W = (\Gamma, \boldsymbol{S}, \boldsymbol{R}, \boldsymbol{C})$ and $W' = (\Gamma', \boldsymbol{S}', \boldsymbol{R}', \boldsymbol{C}')$ be two artifact systems. Then W and W' have the *same basis* if $\Gamma = \Gamma'$. In this case, W and W' are *path-wise equivalent* if the set of W-valid paths is equivalent to the set of W'-valid paths. W and W' are *functionally equivalent* over artifact class C if the set of o-successful paths for W is equal to the set of o-successful paths for W'. They are *functionally equivalent for C-focused paths* for artifact class C if for each $o \in \mathbf{ID}_C$, the set of o-focused, o-successful paths for W is equal to the set of o-focused, o-successful paths for W'

(Obviously, if W and W' are path-wise equivalent, then they are functionally equivalent for each class C.)

Theorem 4.4 Let $W = (\Gamma, \boldsymbol{S}, \boldsymbol{R}, \boldsymbol{C})$ be a monotonic artifact system and C an artifact class in Γ. Then it is Π_2^P-complete whether there is a dead-end path for C in W. This remains true under the various restrictions described in the statement of Theorem 4.3.

We now provide a construction that can be used to eliminate dead-end paths.

Theorem 4.5 Let $W = (\Gamma, \boldsymbol{S}, \boldsymbol{R}, \boldsymbol{C})$ be a monotonic artifact system and C an artifact class in Γ, and C a class in Γ. Then there is an artifact system $W' = (\Gamma, \boldsymbol{S}, \boldsymbol{R}, \boldsymbol{C} \cup \boldsymbol{C}')$, with $\boldsymbol{C}'$ a family of undefined-att-state-blocking constraints, which is functionally equivalent for C-focused paths to W, and for each $o \in \mathbf{ID}_C$, W' has no o-focused dead-end paths. Further, the size of W' is no greater than exponential in the size of W.

A similar result can be obtained that starts with an artifact system with no constraints, and produces a functionally equivalent artifact system with no constraints and no dead-end paths.

Turning to **Q3**, we show that this problem is decidable under the same restrictions.

Theorem 4.6 Let $W = (\Gamma, \boldsymbol{S}, \boldsymbol{R})$ be an artifact system. The problem of deciding whether an attribute A of a class $C \in \Gamma$ is redundant is coNP-complete for all cases (a, b, c, d, e) of Theorem 4.3.

Finally, we briefly outline an extension of the positive results.

Definition. Let Γ be a schema and I, J two instances of Γ. J *link-extends* I, $I \leqslant_{\mathrm{L}} J$, if for each class C, each ID_C o, and each attribute A of C whose type is artifact ID, (1) there is an artifact in I with ID o implies that there an artifact in J with ID o, and (2) $J(o).A = I(o).A$.

Definition. Let $W = (\Gamma, \boldsymbol{S}, \boldsymbol{R})$ be an artifact system, and $k \geqslant 0$. A path $I_0, I_1, ..., I_n$ is a *k-fixed-link structure* if (1) for each $j \in [0..k]$, I_j has at most k artifacts for each class, and (2) for each $j \in [1..n]$, $I_{j-1} \leqslant_{\mathrm{L}} I_j$.

Paths for the Order artifact of Section 2 hav 4-fixed-link structure.

Theorem 4.7 For each k, Theorems 4.2, 4.3, 4.5, and 4.6 hold for k-fixed-link structure paths.

5   Related work

The concept of business artifacts is introduced in [15] and further studied in [3, 11, 6]. In [3], the authors lay out the methodology in the context of Model Driven Business Transformation and describes the positive feedback received in real-world engagements. [11] presents nine patterns emerging in artifact-centric process models and develops a computational model based on Petri Nets. [6] uses a different model and develops static analysis techniques for artifact-centric model properties such as arrival, persistence, and uniqueness.

Many tools and techniques proposed for the development of business process models using workflows (e.g., [9, 14, 10]). These approaches have followed a process-centric approach focused on the control and coordination of tasks [5]. The importance of a data-centric view of processes is also advocated in [2] and [8]. In [2], the authors encourage an "object view" of scientific workflows where the data generated and used is the central focus; while [8] investigates "attribute-centric" workflows where attributes and modules have states. [13] proposes a mixed approach which can express both control and data flow. Compared to these approaches, our work favors a data-centric view.

Another thread of related work is the new paradigm of workflow research which concerns both control flows and data flows. The Product-driven case handling approach [1] addresses many concerns of traditional workflows especially with respect to the treatment of process context or data. Wang and Kumar [20] proposed document-driven workflow systems where data dependencies, in addition to control flows, are introduced into process design in order to make more efficient process design. In their framework, business tasks are defined using input and output documents as well as other constraints, like business rules and policies, imposed on the documents. In comparison, our artifact-centric model re-organizes documents into structured business artifacts, which significantly reduces complexity of modeling data-control flow interactions.

Process verification has been studied extensively in the workflow community, with activity sequencing in Patri nets [19], in graphs [17], data dependencies [18]. (See [6] for additional references.)

6   Conclusions

The artifact-based approach uses key business data, in the form of "artifacts", as the driving force in the design of business processes. It enables a separation of data management concerns from process flow concerns, and can support rich flexibility in the creation and evolution of business processes. In particular, the artifact-based approach holds the promise of enabling automatic creation of new business processes from existing ones, or from initial specifications of the artifacts and basic services for operating on them. This paper lays the foundation for a formal study of the artifact-based approach and its use as the basis for automated workflow creation.

The focus of this paper is on basic decision problems, related to reachability, avoiding dead-ends, and redundancy. While providing key insights, extensions and refinements of these results will be useful, that take into account actual data values, and structural properties of the artifacts and their state diagrams. More broadly, we are interested to develop tools and techniques for automatic construction of business processes, in the spirit of the Semantic Web Services community.

References

1. W.M.P. Aalst, M. Weske, and D. Grnbauer. Case handling: a new paradigm for business process support. *Data and Knowledge Engineering*, 53:129–162, 2005.

2. A. Ailamaki, Y. Ioannidis, and M. Livny. Scientific workflow management by database management. In *Proc. Int. Conf. on Statistical and Scientific Database Management*, 1998.

3. K. Bhattacharya, R. Guttman, K. Lymann, F. F. Heath III, S. Kumaran, P. Nandi, F. Wu, P. Athma, C. Freiberg, L. Johannsen, and A. Staudt. A model-driven approach to industrializing discovery processes in pharmaceutical research. *IBM Systems Journal*, 44(1):145–162, 2005.

4. OWL Services Coalition. OWL-S: Semantic markup for web services, November 2003.

5. D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–154, April 1995.

6. C. E. Gerede and K. Bhattacharya J. Su. Static analysis of business artifact-centric operational models. In *IEEE International Conference on Service-Oriented Computing and Applications*, 2007. to appear.

7. M. Hammer. Deep change: How operational innovation can transform your company. *Havard Business Review*, pages 84–93, April 2004.

8. R. Hull, F. Llirbat, E. Simon, J. Su, G. Dong, B. Kumar, and G. Zhou. Declarative workflows that support easy modification and dynamic browsing. In *Proc. Int. Joint Conf. on Work Activities Coordination and Collaboration*, 1999.

9. M. Jackson and G. Twaddle. *Business Process Implementation Building Workflow Systems*. Addison-Wesley, ACM Press Books, Boston, 1997.

10. F. Leymann and D. Roller. Business process management with flowmark. In *Proc. of COMPCON*, 1994.

11. R. Liu, K. Bhattacharya, and F. Y. Wu. Modeling business contexture and behavior using business artifacts. In *CAiSE*, volume 4495 of *LNCS*, 2007.

12. S. A. McIlraith, T. C. Son, and H. Zeng. Semantic web services. In *IEEE Intelligent Systems*, March/April 2001.
13. C. Medeiros, G. Vossen, and M. Weske. Wasa: a workflow-based architecture to support scientific database applications. In *Proc. 6th DEXA Conference*, 1995.
14. J. P. Morrison. *Flow-Based Programming*. Van Nostrand ReinHold, New York, 1994.
15. A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.
16. R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.
17. Wasim Sadiq and Maria E. Orlowska. Analyzing process models using graph reduction techniques. *Inf. Syst.*, 25(2):117–134, 2000.
18. S. X. Sun, J. F. Nunamaker J. L. Zhao, and O. R. L. Sheng. Formulating the data-flow perspective for business process management. *Information Systems Research*, 17(4):374391, 2006.
19. W. M. P. van der Aalst. The application of Petri nets to workflow management. *J. of Circuits, Systems and Computers*, 8(1), 1998.
20. J. Wang and A. Kumar. A framework for document-driven workflow systems. In *Business Process Management*, pages 285–301, 2005.