# Automatic Verification of Database-Centric Systems[*]

Alin Deutsch
UC San Diego
deutsch@cs.ucsd.edu

Richard Hull
IBM T.J. Watson Research Center
hull@us.ibm.com

Victor Vianu
UC San Diego & INRIA Saclay
vianu@cs.ucsd.edu

## 1. INTRODUCTION

Software systems centered around a database are pervasive in numerous applications. They are encountered in areas as diverse as electronic commerce, e-government, scientific applications, enterprise information systems, and business process management. Such systems are often very complex and prone to costly bugs, whence the need for verification of critical properties.

Classical software verification techniques that can be applied to such systems include *model checking* and *theorem proving*. However, both have serious limitations. Indeed, model checking usually requires performing finite-state abstraction on the data, resulting in loss of semantics for both the system and properties being verified. Theorem proving is incomplete, requiring expert user feedback.

Recently, an alternative approach to verification of database-centric systems has taken shape, at the confluence of the database and computer-aided verification areas. It aims to identify restricted but sufficiently expressive classes of database-driven applications and properties for which sound and complete verification can be performed in a fully automatic way. This approach leverages another trend in database-driven applications: the emergence of high-level specification tools for database-centered systems, such as interactive web applications and data-driven business processes. We review next a few representative examples.

A commercially successful high-level specification tool for web applications is Web Ratio [1], an outgrowth of the earlier academic prototype WebML [20, 17]. Web Ratio allows to specify a Web application using an interactive variant of the E-R model augmented with a workflow formalism. Non-interactive variants of Web page specifications had already been proposed in Strudel [41], Araneus [60] and Weave [42], targeting the automatic generation of Web sites from an underlying database. High-level specification tools have also emerged in the area of business process management, concomitantly with an evolution from the traditional process-centric approach towards data awareness. A notable exponent of this class is the *business artifact model* pioneered in [65, 53], deployed by IBM in professional services. Business artifacts (or simply "artifacts") model key business-relevant entities, which are updated by a set of services that implement business process tasks. A collection of artifacts and services is called an *artifact system*. This modeling approach has been successfully deployed in practice [7, 6, 21, 27, 73], and has been adopted in the OMG standard for Case Management [9].

Tools such as the above automatically generate the database-centric application code from the high-level specification. This not only allows fast prototyping and improves programmer productivity but, as a side effect, provides new opportunities for automatic verification. Indeed, the high-level specification is a natural target for verification, as it addresses the most likely source of errors (the application's specification, as opposed to the less likely errors in the automatic generator's implementation).

The theoretical and practical results obtained so far concerning the verification of such systems are quite encouraging. They suggest that, unlike arbitrary software systems, significant classes of data-driven systems may be amenable to automatic verification. This relies on a novel marriage of database and model checking techniques, and is relevant to both the database and the computer-aided verification communities.

In this article, we describe several models and results on automatic verification of database-driven systems, focusing on temporal properties of their underlying workflows. To streamline the presentation, we focus on verification of business artifacts, and use it as a vehicle to introduce the main con-

---

[*] **Database principles column**. Column editor: Pablo Barceló. Dept. of Computer Science, University of Chile. E-mail: pbarcelo@dcc.uchile.cl.

cepts and results. Moreover, the technical challenges posed by verification of business artifacts are representative of those present in some of the other models (notably data-driven web services), which can be viewed as syntactic variants of business artifacts. We also summarize some of the work pertaining specifically to data-driven web services.

## 2. BUSINESS ARTIFACTS

IBM's business artifacts model key business-relevant entities, which are updated by a set of services that implement business process tasks. The notion of business artifact was first introduced in [65] and [53] (called there "adaptive documents"), and was further studied, from both practical and theoretical perspectives, in [6, 43, 44, 8, 56, 25, 47, 4]. (Some of these publications use the term "business entity" in place of "business artifact"). Some key roots of the artifact model are present in "adaptive business objects" [63], "business entities", "document-driven" workflow [72] and "document" engineering [45]. The Vortex framework [49, 38, 48] also allows the specification of database manipulations and provides declarative specifications for when services are applicable to a given artifact.

The artifact model is inspired in part by the field of semantic web services. In particular, the OWL-S proposal [59, 58] describes the semantics of services in terms of input parameters, output parameters, pre- and post-conditions. In the artifact model considered here the services are applied in a sequential fashion (there is no true concurrency). IBM has developed Siena [23], a tool for compiling artifact-based procedural specifications into code supporting the corresponding business process. Its open-source descendant, the BizArtifact suite [10], has just been announced. The Guard-Stage-Milestone (GSM) approach [25, 47] to artifact lifecycles permits services with pre- and post-conditions, parallelism, and hierarchy. The OMG standard for Case Management Model and Notation (CMMN) [9], announced last year, draws key foundational elements from GSM [57].

We next describe a minimalistic variant of the artifact model, adequate for illustrating the results on verification. The presentation is informal, relying mainly on a running example (the formal development is provided in [30, 24]). The example, modeling an e-commerce process, features several characteristics.

1. The system routinely queries an underlying database, for instance to look up the price of a product and the shipping weight restrictions.

2. The validity checks and updates carried out by the services involve arithmetic operations. For instance, to be valid, an order must satisfy such conditions as: (a) the product weight must be within the selected shipment method's limit, and (b) if the buyer uses a coupon, the sum of product price and shipping cost must exceed the coupon's minimum purchase limit.

3. Finally, the correctness of the business process relies on database integrity constraints. For instance, the system must check that a selected triple of product, shipment type and coupon are globally compatible. This check is implemented by several local tests, each running at a distinct instant of the interaction, as user selections become available. Each local test accesses distinct tables in the database, yet they globally refer to the same product, due to the keys and foreign keys satisfied by these tables.

The example models an e-commerce business process in which the customer chooses a product and a shipment method and applies various kinds of coupons to the order. There are two kinds of coupons: discount coupons subtract their value from the total (e.g. a $50 coupon) and free-shipment coupons subtract the shipping costs from the total. The order is filled in a sequential manner (first pick the product, then the shipment, then claim a coupon), as is customary on e-commerce web-sites. After the order is filled, the system awaits for the customer to submit a payment. If the payment matches the amount owed, the system proceeds to shipping the product.

As mentioned earlier, an artifact is an evolving record of values. The values are referred to by variables (sometimes called *attributes*). In general, an artifact system consists of several artifacts, evolving under the action of *services*, specified by pre- and post-conditions. This notion of service corresponds roughly to the notion of task in BPM (although it may also cover web services implementing tasks). For instance, in GSM terms, a service corresponds to a *stage*, a pre-condition to a *guard*, and a post-condition to a *milestone*.

In the example, we use a single artifact with the following variables:

`status`,`prod_id`,`ship_type`,`coupon`
`amount_owed` `amount_paid`,`amount_refunded`

The `status` variable tracks the status of the order and can take the following values:

"edit_product", "edit_ship", "edit_coupon"
"processing", "received_payment",
"shipping", "shipped", "canceling", "canceled".

Artifact variables `ship_type` and `coupon` record the customer's selection, received as an external input. `amount_paid` is also an external input (from the customer, possibly indirectly via a credit card service). Variable `amount_owed` is set by the system using arithmetic operations that sum up product price and shipment cost, subtracting the coupon value. Variable `amount_refunded` is set by the system in case a refund is activated.

The database includes the following tables, where underlined attributes denote keys. Recall that a key is a set of attributes that uniquely identify each tuple in a relation.

$\mathtt{PRODUCTS}(\underline{\mathtt{id}}, \mathtt{price}, \mathtt{availability}, \mathtt{weight})$
$\mathtt{COUPONS}(\underline{\mathtt{code}}, \mathtt{type}, \mathtt{value}, \mathtt{min\_value}, \mathtt{free\_shiptype})$
$\mathtt{SHIPPING}(\underline{\mathtt{type}}, \mathtt{cost}, \mathtt{max\_weight})$
$\mathtt{OFFERS}(\underline{\mathtt{prod\_id}}, \mathtt{discounted\_price}, \mathtt{active})$

The database also satisfies the following foreign keys:

$\mathtt{COUPONS}[\mathtt{free\_shiptype}] \subseteq \mathtt{SHIPPING}[\mathtt{type}]$ and $\mathtt{OFFERS}[\mathtt{prod\_id}] \subseteq \mathtt{PRODUCTS}[\mathtt{id}]$.

The first inclusion dependency says that each `free_shiptype` value in the `COUPONS` relation is also a `type` value in the `SHIPPING` relation. The second dependency states that every `prod_id` value in the `OFFERS` is the actual `id` of a product in the `PRODUCTS` relation.

The starting configuration of every artifact system is constrained by an initialization condition, which here states that `status` is initialized to "edit_prod", and all other variables to "undefined". By convention, we model undefined variables using the reserved constant $\lambda$.

**The services.** Recall that artifacts evolve under the action of services. As in the Guard-Stage Milestone approach mentioned above, each service is specified declaratively by a pre-condition $\pi$ and a post-condition $\psi$, here limited to existential first-order ($\exists$FO) sentences. The pre-condition refers to the current values of the artifact variables and the database. The post-condition $\psi$ refers simultaneously to the current and *next* artifact values, as well as the database. In addition, both $\pi$ and $\psi$ may use arithmetic constraints on the variables, limited to linear inequalities over the rationals.

The services shown in Figure 1 model a few of the business process tasks of the example. Throughout the example, we use primed artifact variables $x'$ to refer to the *next* value of variable $x$.

Notice that the pre-conditions of the services check the value of the `status` variable. For instance, according to **choose_product**, the customer can only input her product choice while the order is in "edit_prod" status.

Also notice that the post-conditions constrain the next values of the artifact variables (denoted by a prime). For instance, according to **choose_product**, once a product has been picked, the next value of the status variable is "edit_shiptype", which will at a subsequent step enable the **choose_shiptype** service (by satisfying its pre-condition). Similarly, once the shipment type is chosen (as modeled by service **choose_shiptype**), the new status is "edit_coupon", which enables the **apply_coupon** service. The interplay of pre- and post-conditions achieves a sequential filling of the order, starting from the choice of product and ending with the claim of a coupon.

A post-condition may refer to both the current and next values of the artifact variables. For instance, in service **choose_shiptype**, the fact that only the shipment type is picked while the product remains unchanged, is modeled by preserving the product id: the next and current values of the corresponding artifact variable are set equal.

Pre- and post-conditions may query the database. For instance, in service **choose_product**, the post-condition ensures that the product id chosen by the customer is that of an available product (by checking that it appears in a `PRODUCTS` tuple, whose availability attribute is positive).

Finally, notice the arithmetic computation in the post-conditions. For instance, in service **apply_coupon**, the sum of the product price $p$ and shipment cost $c$ (looked up in the database) is adjusted with the coupon value (notice the distinct treatment of the two coupon types) and stored in the `amount_owed` artifact variable.

Observe that the first post-condition disjunct models the case when the customer inputs no coupon number (the next value $\mathtt{coupon}'$ is set to undefined), in which case a different owed amount is computed, namely the sum of price and shipping cost.

**Semantics** The semantics of an artifact system $\mathcal{A}$ consists of its *runs*. Given a database $D$, a run of $\mathcal{A}$ is an infinite sequence $\{\rho_i\}_{\geq 0}$ of artifact records such that $\rho_0$ and $D$ satisfy the initial condition of the system, and for each $i \geq 0$ there is a service $S$ of the system such that $\rho_i$ and $D$ satisfy the pre-condition of $S$ and $\rho_i$, $\rho_{i+1}$ and $D$ satisfy its post-condition. For uniformity, blocking prefixes of runs are extended to infinite runs by repeating forever their last record.

**choose_product**: The customer chooses a product.
$\pi : \texttt{status} = \text{“edit\_prod”}$
$\psi : \exists p, a, w(\texttt{PRODUCTS}(\texttt{prod\_id}', p, a, w) \wedge a > 0) \wedge \texttt{status}' = \text{”edit\_shiptype”}$
**choose_shiptype**: The customer chooses a shipping option.
$\pi : \texttt{status} = \text{“edit\_ship”}$
$\psi : \exists c, l, p, a, w(\texttt{SHIPPING}(\texttt{ship\_type}', c, l) \wedge \texttt{PRODUCTS}(\texttt{prod\_id}, p, a, w) \wedge l > w) \wedge$
$\texttt{status}' = \text{“edit\_coupon”} \wedge \texttt{prod\_id}' = \texttt{prod\_id}$
**apply_coupon**: The customer optionally inputs a coupon number.
$\pi : \texttt{status} = \text{“edit\_coupon”}$
$\psi : (\texttt{coupon}' = \lambda \wedge \exists p, a, w, c, l(\texttt{PRODUCTS}(\texttt{prod\_id}, p, a, w) \wedge$
$\texttt{SHIPPING}(\texttt{ship\_type}, c, l) \wedge \texttt{amount\_owed}' = p + c) \wedge \texttt{status}' = \text{“processing”}$
$\wedge \texttt{prod\_id}' = \texttt{prod\_id} \wedge \texttt{ship\_type}' = \texttt{ship\_type}) \vee$
$(\exists t, v, m, s, p, a, w, c, l(\texttt{COUPONS}(\texttt{coupon}', t, v, m, s) \wedge$
$\texttt{PRODUCTS}(\texttt{prod\_id}, p, a, w) \wedge \texttt{SHIPPING}(\texttt{ship\_type}, c, l) \wedge p + c \geq m \wedge$
$(t = \text{“free\_shipping”} \rightarrow (s = \texttt{ship\_type} \wedge \texttt{amount\_owed}' = p)) \wedge$
$(t = \text{“discount”} \rightarrow \texttt{amount\_owed}' = p + c - v))$
$\wedge \texttt{status}' = \text{“processing”} \wedge \texttt{prod\_id}' = \texttt{prod\_id} \wedge \texttt{ship\_type}' = \texttt{ship\_type})$

**Figure 1: Three services**

Note that the above semantics only considers linear runs of the system. A more informative notion is the *tree of runs* that completely captures the choice of services applicable at any given stage in the computation. We confine ourselves to linear runs because we are interested in verifying linear-time properties of the system. Formulating and verifying branching-time properties would require a semantics consisting of the full tree of runs.

The business process in the example exhibits a flexibility that, while desirable in practice for a postive customer experience, yields intricate runs, all of which need to be considered in verification. For instance, at any time before submitting a valid payment, the customer may edit the order (select a different product, shipping method, or change/add a coupon) an unbounded number of times. Likewise, the customer may cancel an order for a refund even after submitting a valid payment.

## 3. SPECIFYING TEMPORAL PROPERTIES OF DATA-CENTRIC SYSTEMS

We are interested in verifying temporal properties of runs of data-centric systems such as business artifacts. For instance, in our artifact system example, we would like to express such desiderata as:

> If a correct payment is submitted then at some time in the future either the product is shipped or the customer is refunded the correct amount.
>
> A free shipment coupon is accepted only if the available quantity of the product is greater than zero, the weight of the product is in the limit allowed by the shipment method, and the sum of price and shipping cost exceeds the coupon's minimum purchase value.

Similar properties are of interest for the data-driven web services described in Section 5. In order to specify such temporal properties we use an extension of LTL (linear-time temporal logic). Recall that LTL is propositional logic augmented with temporal operators such as **G** (always), **F** (eventually), **X** (next) and **U** (until) (e.g., see [66]). For example, $\mathbf{G}p$ says that $p$ holds at all times in the run, $\mathbf{F}p$ says that $p$ will eventually hold, and $\mathbf{G}(p \rightarrow \mathbf{F}q)$ says that whenever $p$ holds, $q$ must hold sometime in the future. The extension of LTL that we use, called[1] LTL-FO, is obtained from LTL by replacing propositions with quantifier-free FO statements about particular artifact records in the run. The statements use the artifact variables and may use additional *global* variables, shared by different statements and allowing to refer to values in different records. The global variables are universally quantified over the entire property.

For example, suppose we wish to specify the property that if a correct payment is submitted then at some time in the future either the product is shipped or the customer is refunded the correct amount. The property is of the form $\mathbf{G}(p \rightarrow \mathbf{F}q)$,

---

[1] The variant of LTL-FO used here differs from previous ones in that the FO formulas interpreting propositions are quantifier-free. By slight abuse we use here the same name.

where $p$ says that a correct payment is submitted and $q$ states that either the product is shipped or the customer is refunded the correct amount. Moreover, if the customer is refunded, the amount of the correct payment (given in $p$) should be the same as the amount of the refund (given in $q$). This requires using a global variable $x$ in both $p$ and $q$. More precisely, $p$ is interpreted as the formula $\texttt{amount\_paid} = x \wedge \texttt{amount\_paid} = \texttt{amount\_owed}$ and $q$ as $\texttt{status} = \text{"shipped"} \vee \texttt{amount\_refunded} = x$. This yields the LTL-FO property

$$
\begin{aligned}
&(\varphi_1)\ \ \forall x \mathbf{G}((\texttt{amount\_paid} = x \\
&\wedge \texttt{amount\_paid} = \texttt{amount\_owed})\ \rightarrow \\
&\mathbf{F}(\texttt{status} = \text{"shipped"} \vee \texttt{amount\_refunded} = x))
\end{aligned}
$$

Note that, as one would expect, the global variable $x$ is universally quantified at the end. We say that an artifact system $\mathcal{A}$ satisfies an LTL-FO sentence $\varphi$ if all runs of the artifact system satisfy $\varphi$ for all values of the global variables. Note that the database is fixed for each run, but may be different for different runs.

We now show a second property $\varphi_2$ for the running example, expressed by the LTL-FO formula

$$
\begin{aligned}
&(\varphi_2)\ \ \forall v, m, s, p, a, w, c, l \\
&(\mathbf{G}((\texttt{prod\_id} \neq \lambda \wedge\ \ \texttt{ship\_type} \neq \lambda\ \ \wedge \\
&\texttt{COUPONS}(\texttt{coupon}, \text{"free\_ship"}, v, m, s)\ \ \wedge \\
&\texttt{PRODUCTS}(\texttt{prod\_id}, p, a, w)\ \ \wedge \\
&\texttt{SHIPPING}(\texttt{ship\_type}, c, l)) \rightarrow \\
&(\underbrace{a > 0}_{(i)}\ \ \wedge\ \ \underbrace{w \leq l}_{(ii)} \wedge\ \ \underbrace{p + c \geq m}_{(iii)})))
\end{aligned}
$$

Property $\varphi_2$ verifies the consistency of orders that use coupons for free shipping. The premise of the implication lists the conditions for a completely specified order that uses such coupons. The conclusion checks the following business rules: (i) available quantity of the product is greater than zero, (ii) the weight of the product is in the limit allowed by the shipment method, and (iii) the total order value satifies the minimum for the application of the coupon.

We note that variants of LTL-FO have been introduced in [39, 70]. The use of globally quantified variables is also similar in spirit to the *freeze quantifier* defined in the context of LTL extensions with data by Demri and Lazić [28, 29].

### *Other applications of verification*

As discussed in [30], various useful static analysis problems on business artifacts can be reduced to verification of temporal properties. We mention some of them.

**Business rules** The basic artifact model is extended in [8] with *business rules*, in order to support service reuse and customization. Business rules are conditions that can be super-imposed on the pre-conditions of existing services without changing their implementation. They are useful in practice when services are provided by autonomous third-parties, who typically strive for wide applicability and impose as unrestrictive pre-conditions as possible. When such third-party services are incorporated into a specific business process, this often requires more control over when services apply, in the form of more restrictive pre-conditions. Such additional control may also be needed to ensure compliance with business regulations formulated by third parties, independently of the specific application. Verification of properties in the presence of business rules then becomes of interest and can be addressed by our techniques. A related issue is the detection of *redundant* business rules, that do not affect the runs of the system. This can also be reduced to a verification problem.

**Redundant attributes** Another design simplification consists of redundant attribute removal, a problem also raised in [8]. This is formulated as follows. We would like to test whether there is a way to satisfy a property $\varphi$ of runs without using one of the attributes. This easily reduces to a verification problem as well.

**Runtime analysis** The verification techniques described above can also be used to perform useful runtime analysis tasks. Examples include providing guidance to users trying to achieve certain goals, runtime monitoring of events, what-if scenarios, and diagnosis of anomalous behavior based on partial traces of an artifact execution (e.g. [2]). This is also in the spirit of the line of research on runtime operational support in BPM [71].

## 4. AUTOMATIC VERIFICATION OF ARTIFACT SYSTEMS

Classical model checking applies to finite-state transition systems. While finite-state systems may fully capture the semantics of some systems to be verified (for example logical circuits), most software systems are in fact infinite-state systems, of which a finite-state transition system represents a rough abstraction. Properties of the actual system are also abstracted, using a finite set of propositions whose truth values describe each of the finite states of the transition system. Checking that an LTL property holds is done by searching for a counterexample run of the system. Its finiteness is essential and allows to decide property satisfaction in PSPACE using an automata-theoretic approach (see e.g. [22, 61]).

Consider now an artifact system $\mathcal{A}$ and an LTL-FO property $\varphi$. Model checking $\mathcal{A}$ with respect to $\varphi$ can be viewed once again as a search for a counterexample run of $\mathcal{A}$, i.e. a run violating $\varphi$. The immediate difficulty, compared to the classical approach, stems from the fact that $\mathcal{T}_\mathcal{A}$ is an infinite-state system. To obtain decidability in this context, the typical approach consists of using *symbolic representations* of runs, as described later.

In the broader context of verification, research on automatic verification of infinite-state systems has also focused on extending classical model checking techniques (e.g., see [18] for a survey). However, in much of this work the emphasis is on studying recursive control rather than data, which is either ignored or finitely abstracted. More recent work has been focusing specifically on data as a source of infinity. This includes augmenting recursive procedures with integer parameters [13], rewriting systems with data [14, 12], Petri nets with data associated to tokens [54], automata and logics over infinite alphabets [16, 15, 64, 28, 52, 11, 12], and temporal logics manipulating data [28, 29]. However, the restricted use of data and the particular properties verified have limited applicability to the business artifact setting, or other database-driven applications.

### *Artifacts without constraints or dependencies*

We consider first artifact systems and properties without arithmetic constraints or data dependencies. This case was studied in [30], with a slightly richer model in which artifacts can carry some limited relational state information (however, here we stick for simplicity to the earlier minimalistic model). The main result is the following.

THEOREM 4.1. *It is decidable, given an artifact system $\mathcal{A}$ with no data dependencies or arithmetic constraints, and an LTL-FO property $\varphi$ with no arithmetic constraints, whether $\mathcal{A}$ satisfies $\varphi$.*

The complexity of verification is PSPACE-complete for fixed-arity database and artifacts, and EXPSPACE otherwise. This is the best one can expect, given that even very simple static analysis problems for finite-state systems are already PSPACE-complete [68].

The main idea behind the verification algorithm is to explore the space of runs of the artifact system using *symbolic* runs rather than actual runs. This is based on the fact that the relevant information at each instant is the pattern of connections in the database between attribute values of the current and successor artifact records in the run, referred to as their *isomorphism type*. Indeed, the sequence of isomorphism types in a run can be generated symbolically and is enough to determine satisfaction of the property. Since each isomorphism type can be represented by a polynomial number of tuples (for fixed arity), this yields PSPACE verification.

It turns out that the verification algorithm can be extended to specifications and properties that use a *total order* on the data domain, which is useful in many cases. This however complicates the algorithm considerably, since the order imposes global constraints that are not captured by the local isomorphism types. The algorithm was first extended in [30] for the case of a dense countable order with no end-points. This was later generalized to an arbitrary total order by Segoufin and Torunczyk [67] using automata-theoretic techniques. In both cases, the worst-case complexity remains PSPACE.

### *Artifacts with arithmetic constraints and data dependencies*

Unfortunately, Theorem 4.1 fails even in the presence of simple data dependencies or arithmetic. Specifically, as shown in [30, 24], verification becomes undecidable as soon as the database is equipped with at least one key dependency, *or* if the specification of the artifact system uses simple arithmetic constraints allowing to increment and decrement by one the value of some atributes. Hence, a restriction is needed to achieve decidability. We discuss this next.

To gain some intuition, consider the undecidability of verification for artifact systems with increments and decrements. The proof of undecidability is based on the ability of such systems to simulate *counter machines*, for which the problem of state reachability is known to be undecidable [62]. To simulate counter machines, an artifact system uses an attribute for each counter. A service performs an increment (or decrement) operations by "feeding back" the incremented (or decremented) value into the next occurrence of the corresponding attribute. To simulate counters, this must be done an unbounded number of times. To prevent such computations, the restriction imposed in [24] is designed to limit the data flow between occurrences of the same artifact attribute at different times in runs of the system that satisfy the desired property. As a first cut, a possible restriction would prevent any data flow path between unequal occurrences of the same artifact attribute. Let us call this restriction *acyclicity*. While acyclicity would achieve the goal of rendering verification decidable, it is too strong for many practical situations. In our running example, a customer can choose a shipping type and coupon and repeatedly change her mind and start over. Such repeated performance of a task is useful in many scenarios, but would be prohibited by acyclicity of the data flow. To this end, we define in [24] a more permissive restriction called *feedback freedom*. The formal definition considers, for each run, a graph capturing the data flow among variables, and imposes a restriction on the graph. Intuitively, paths among different occurrences of the same attribute are permitted, but only as long as each value of the attribute is independent on its previous values. This is ensured by a syntactic condition that takes into account both the artifact system and the property to be verified. We omit here the rather technical details. It is shown in [24] that feedback freedom of an artifact system together with an LTL-FO property can be checked in PSPACE by reduction to a test of emptiness of a two-way alternating finite-state automaton. More significantly, artifact systems designed in a hierarchical fashion by successive refinement, in the style of the Guard-Stage-Milestone approach [25, 47], naturally satisfy feedback freedom. Indeed, there is evidence that the feedback freedom condition is permissive enough to capture a wide class of applications of practical interest. This is confirmed by numerous examples of practical business processes modeled as artifact systems. Many of these, including typical e-commerce applications, satisfy the feedback freedom condition. Feedback freedom turns out to ensure decidability of verification in the presence of arithmetic constraints, and also under a large class of data dependencies including key and foreign key constraints on the database.

THEOREM 4.2. *[24] It is decidable, given an artifact system $\mathcal{A}$ whose database satisfies a set of key and foreign key constraints, and an LTL-FO property $\varphi$ such that $(\mathcal{A}, \varphi)$ is feedback free, whether every run of $\mathcal{A}$ on a valid database satisfies $\varphi$.*

The intuition behind decidability is the following. Recall the verification algorithm of Theorem 4.1. Because of the data dependencies and arithmetic constraints, the isomorphism types of symbolic runs no longer suffice, because every artifact record in a run is constrained by the entire history leading up to it. This can be specified as an $\exists$FO formula using one quantified variable for each artifact attribute occurring in the history, referred to as the *inherited constraint* of the record. The key observation is that due to feedback freedom, the inherited constraint can be rewritten into an $\exists$FO formula with quantifier rank bounded by $k^2$, where $k$ is the number of attributes of the artifact (the quantifier rank of a formula is the maximum number of quantifiers occurring along a path from root to leaf in the syntax tree of the formula, see [55]). This implies that there are only finitely many non-equivalent inherited constraints. This allows to use again a symbolic run approach to verification, by replacing isomorphism types with inherited constraints.

One might wonder if the positive results of this section can be extended to branching-time logics (CTL or CTL*). Unfortunately, it is easily shown that even very simple CTL properties become undecidable in the present framework. It remains open whether there are reasonable restrictions that guarantee decidability of CTL or CTL*. We note that limited positive results on verification of branching-time properties of data-driven web services are obtained in [34].

### *Other work on verification of artifact systems*

Recently, a line of work on automatic verification of database-centric business processes (specified using formalisms isomorphic to artifact systems) has introduced a variant of the verification problem in which properties are checked only over the runs starting from a given initial database. During the run, the database may evolve via updates, insertions and deletions. In particular, it may be extended with fresh values provided as input throughout the run. Since inputs come from an infinite domain, this verification variant remains infinite-state. The property languages are fragments of first-order-extended $\mu$-calculus [26]. Decidability results in this context are based on sufficient syntactic restrictions. One such restriction ensures that the number of fresh input values is bounded throughout every run [26, 46]. The restriction exploits an analogy between artifact system runs and sequences of chase steps with embedded dependencies, and it corresponds to the notion of "weakly acyclic" set of dependencies [40]. A complementary type of restriction allows an unbounded number of distinct inputs during the run, but not their unbounded accumulation within the database, implying a bound on the latter's size. This restriction, called "generate-recall acyclicity", is based on a data flow analysis of how cyclic generation of fresh inputs interacts with their cyclic storage (recall) during the run [46]. [5] derives decidability of the verification variant by also disallowing unbounded accumulation of input values, but this condition is postulated as a semantic property (shown undecidable in [46]).

Additional results on formal analysis of artifact-centric business processes in restricted contexts have been reported in [43, 44, 8]. Properties investigated in these studies include reachability [43, 44], general temporal constraints [44], and the existence of complete execution or dead end [8]. Citations [43, 44] are focused on an essentially procedural version of artifact-centric workflow, and [8] is the first to study a declarative version. For the variants considered in each paper, verification is generally undecidable; decidability results were obtained when rather severe restrictions are placed, e.g., restricting all guards on state transitions to be "true" [43], restricting to bounded domains [44, 8], or restricting the language for conditions to refer only to artifacts (and not their attribute values) [44]. None of the above papers permits an arbitrary global database, separate from the artifacts. See [51] for a survey on data-centric business process management, and [19] for a survey of corresponding verification results.

## 5. DATA-DRIVEN WEB SERVICES

The goal of the Web services paradigm is to enable the use of Web-hosted services with a high degree of flexibility and reliability. Web services can function in a stand-alone manner, or they can be "glued" together into multi-peer *compositions* that implement complex applications. To describe and reason about Web services, various standards and models have been proposed, focusing on different levels of abstraction and targeting different aspects of the Web service. We refer to [50] for a tutorial.

We illustrate with an example the WebML approach to specifying data-driven web services, formally studied in [33, 35].

Consider the common scenario of a web service[2] that takes input from external users and responds by producing output. The contents of a Web page is determined dynamically by querying the underlying database as well as the state. The output of the Web site, transitions from one Web page to another, and state updates, are determined by the current input, state, and database, and defined by first-order queries. We are interested in services specified by a high-level tool such as WebML (and Web Ratio).

We illustrate in Figure 2 a WebML-style specification of an e-commerce Web site selling computers online. New customers can register a name and password, while returning customers can login, search for computers fulfilling certain criteria, add the results to a shopping cart, and finally buy the items in the shopping cart.

A run of the above Web site starts as follows. Customers begin at the home page by providing their login name and password, and choosing one of the provided buttons (login, register, or cancel). Suppose the choice is to login. The reaction of the Web site is determined by a query checking if the name and password provided are found in the database of registered users. If the answer is positive, the login is successful and the customer proceeds to the Customer page or the Administration page depending on his status. Otherwise, there is a transition to the Error page. This continues as described by the flowchart in the figure.

### *Verification of data-driven web services*

The verification problem for database-driven web services has been studied using a transducer-based formal model, called *Extended Abstract State Machine Transducer*, in brief ASM$^+$ . The transducer model captures in a simple way the essential features of relational database-driven reactive systems. The model is an extension of the Abstract State Machine (ASM) transducer previously studied by Spielmann [70]. Similarly to the earlier Relational Transducers of Abiteboul et. al. [3], ASM$^+$ transducers model database-driven reactive systems that respond to input events by producing some output, and maintain state information in designated relations. The control of the device is specified using first-order queries. The main motivation for ASM$^+$ transducers is that they are sufficiently powerful to simulate complex Web service specifications in the style of WebML. Thus, they are a convenient vehicle for developing the theoretical foundation for the verification of such systems, and they also provide the basis for the implementation of a verifier.

As in the case of business artifacts, restrictions are needed on the ASM$^+$ transducers and properties in order to ensure decidability of verification. The main restriction, first proposed in [70] for ASM transducers, is called "input boundedness". The core idea of input boundedness is that quantifications used in formulas of the specification and property are guarded by input atoms. For example, if *pay* is an input, the LTL-FO formula (where **B** is shorthand for *before*)

$$\forall x\ (\mathbf{G}\ (\exists z(pay(x,z) \wedge price(x,z))\ \mathbf{B}\ ship(x)))$$

is input bounded, since the quantification $\exists z$ is guarded by $pay(x,z)$. This restriction matches naturally the intuition that the system modeled by the transducer is input driven. The actual restriction is quite technical, but provides an appealing package. First, it turns out to be tight, in the sense that

---

[2] We interpret "web service" broadly to include SOA-style services as well as web applications and web sites. Although artifact services are different, they may in fact be implemented by web services.

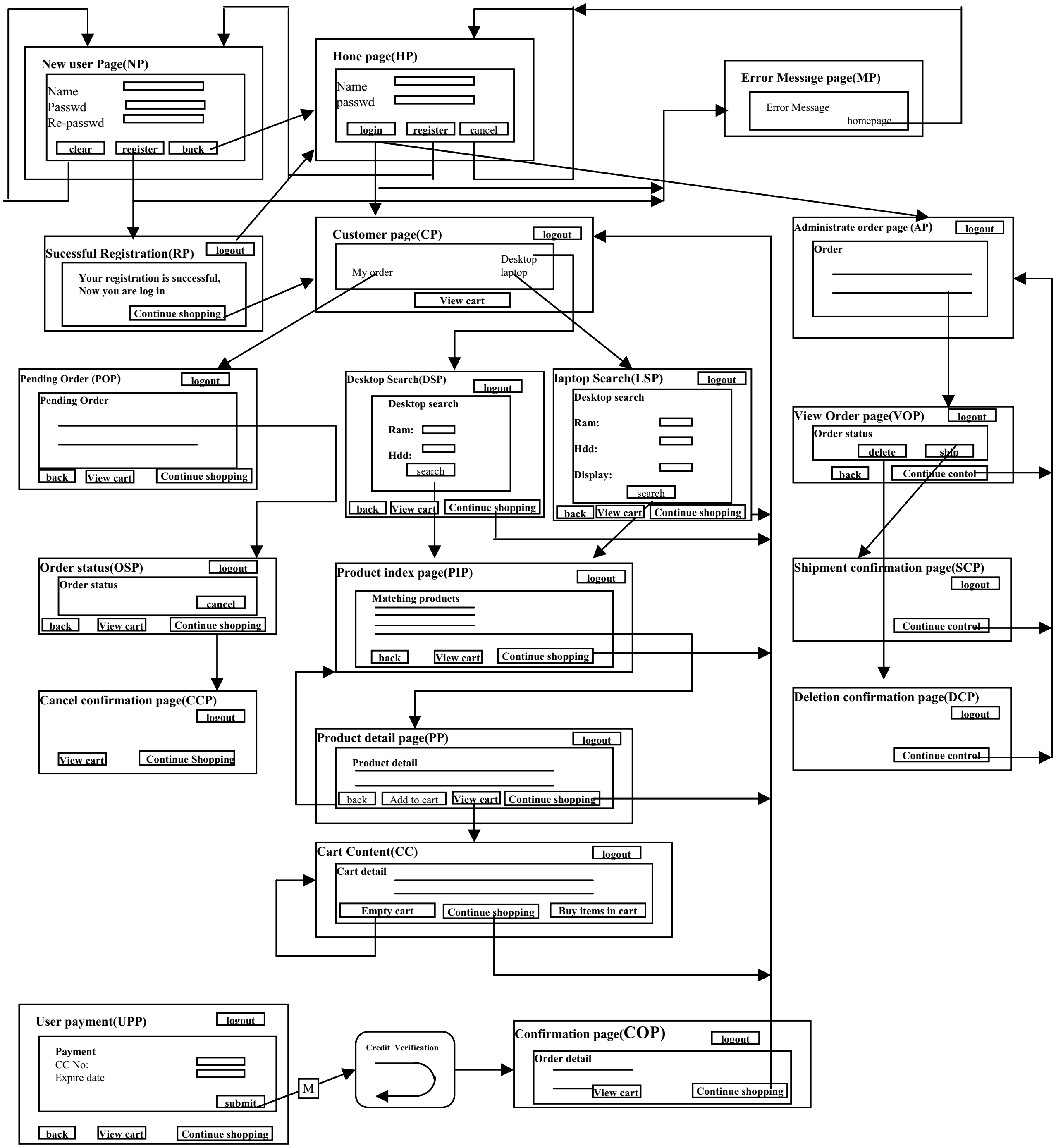

Figure 2: Web pages in the computer shopping site.

even small relaxations lead to undecidability. Second, as argued in [33, 35], it remains sufficiently rich to express a significant class of practically relevant applications and properties. As a typical example, the e-commerce Web application illustrated in Figure 2 can be modeled under this restriction, and many relevant natural properties can be expressed. Third, as in the case of artifacts without dependencies or arithmetic, the complexity of verification is PSPACE (for fixed-arity schemas). Moreover, the proof technique developed to show decidability in PSPACE provides the basis for the implementation of an actual verifier, described next.

### *The WAVE Verifier*

While the PSPACE upper bound obtained for verification in the input-bounded case is encouraging from a theoretical viewpoint, it does not provide any indication of practical feasibility. Fortunately, it turns out that the symbolic approach described above also provides a good basis for efficient implementation. Indeed, this technique lies at the core of the WAVE verifier, targeted at data-driven Web services of the WebML flavor [37, 32].

The verifier, as well as its target specification framework, are both implemented from scratch. Thus, we first developed a tool for high-level, efficient specification of data-driven Web services, in the spirit of WebML. Next, we implemented WAVE taking as input a specification of a Web service using our tool, and an LTL-FO property to be verified. The starting point for the implementation is the symbolic run technique. Indeed, the verifier basically carries out a search for counterexample symbolic runs. However, verification becomes practical only in conjunction with an array of additional heuristics and optimization techniques, yielding critical improvements. Chief among these is dataflow analysis, allowing to dramatically prune the search for counterexample runs.

The verifier was evaluated on a set of practically significant Web application specifications, mimicking the core features of sites such as Dell, Expedia, and Barnes and Noble. The experimental results are quite exciting: we obtained surprisingly good verification times (on the order of seconds), suggesting that automatic verification is practically feasible for significant classes of properties and Web services. The implementation and experimental results are described in [32], and a demo of the WAVE prototype was presented in [36].

### *Compositions of ASM$^+$ Transducers*

The verification results discussed above apply to single ASM$^+$ transducers in isolation. These results were extended in [37] to the more challenging but practically interesting case of *compositions* of ASM$^+$ transducers, modeling compositions of database-driven Web services. Asynchronous communication between transducers adds another dimension that has to be taken into account. In an ASM$^+$ composition, the transducers communicate with each other by sending and receiving messages via one-way channels. Properties of runs to be verified are specified in an extension of LTL-FO, where the FO components may additionally refer to the messages currently read and received.

Towards decidable verification, we extend in a natural way the input-boundedness restriction. Additional restrictions must be placed on the message channels: they may be lossy, but are required to be bounded. With these restrictions, verification is again shown to be PSPACE-complete (for fixed-arity relations, and EXPSPACE otherwise). The proof is by reduction to the single transducer case, and the restrictions are shown to be tight.

The above model of compositions assumes that all specifications of participating peers are available to the verifier. However, compositions may also involve autonomous parties unwilling to disclose the internal implementation details. In this case, the only information available is typically a specification of their input-output behavior. This leads to an investigation of *modular* verification. It consists in verifying that a subset of fully specified transducers behaves correctly, subject to input-output properties of the other transducers. Decidability results are obtained in [37] for verification, subject to an appropriate extension of the input-boundedness restriction.

## 6. CONCLUSIONS

Database-driven systems provide the backbone of many complex applications for which verification is critically important. A fortunate development facilitating this task is the emergence of high-level specification tools centered around database queries, that provide a natural target for verification. The results we described suggest that verification may indeed be feasible for significant classes of database-driven systems so specified. Specifically, we considered two representative models, business artifacts and data-driven web services, and identified restrictions guaranteeing decidability of verification. However, more work is needed at several levels in order to improve the applicability and impact of the results.

On the theoretical front, the quest for the right package of restrictions that enable verification while capturing more relevant sets of specifications is still ongoing. For example, real-life artifacts often contain data beyond flat tuples, such as lists or sets (think of a shopping cart). While unrestricted use of sets is known to lead to undecidability of verification [30], preliminary results [31] suggest that carefully limited use of sets might preserve decidability, while allowing to model common use cases such as the shopping cart. Other useful extensions involve checking properties of the interaction among multiple actors in the workflow, or among multiple artifact instances evolving in parallel. The interoperation, evolution, and integration of multiple workflows also raise important static analysis questions.

Bridging the gap between the abstract setting of theoretical results and full-fledged specification frameworks also raises significant challenges. The decidability results are subject to strong restrictions. The boundary of decidability is subtle, as even small deviations from the restrictions may lead to undecidability. This raises a need to provide tools to guide the design of full-fledged specifications towards satisfaction of the restrictions, whenever possible (e.g. see [69]).

Clearly, a practical verifier needs to also deal with specifications that do not obey the restrictions needed for decidability. As typical in software verification, this can be done by abstracting the given specification to one that satisfies the restrictions, and verifying the resulting abstraction. For example, if certain arithmetic operations are not supported by the verifier, they can be abstracted as black-box relations, ignoring their semantics. The resulting verifier is guaranteed to be *sound* (it is never wrong when it claims correctness of a specification), but is possibly not *complete* (it may produce false negatives, i.e. candidate counterexamples to the desired property, which need to be validated by the user). The technical challenge lies in automatically generating the abstraction such that it gives up only as little completeness as necessary for decidability.

While the theoretical complexity results provide basic information on the difficulty of verification, its practical feasibility can only be demonstrated by actual implementations. The surprisingly good performance of the implemented WAVE verifier [32] is therefore particularly encouraging. Like the theoretical results, this is made possible by a novel coupling of database and model checking techniques. This suggests that the approach to verification described here is quite promising, and may be just the starting point of a fruitful marriage between the database and computer-aided verification areas.

## 7. REFERENCES

[1] Web Ratio. http://www.webratio.com/.

[2] S. Abiteboul and V. Vianu. Collaborative data-driven workflows: think global, act local. In *PODS*, 2013.

[3] S. Abiteboul, V. Vianu, B. Fordham, and Y. Yesha. Relational transducers for electronic commerce. *JCSS*, 61(2):236–269, 2000.

[4] B.B.Hariri, D.Calvanese, G. D. Giacomo, R. D. Masellis, and P.Felli. Foundations of relational artifacts verification. In *BPM*, 2011.

[5] F. Belardinelli, A. Lomuscio, and F. Patrizi. Verification of gsm-based artifact-centric systems through finite abstraction. In *ICSOC*, 2012.

[6] K. Bhattacharya, N. S. Caswell, S. Kumaran, A. Nigam, and F. Y. Wu. Artifact-centered operational modeling: Lessons from customer engagements. *IBM Sys. Journal*, 46(4), 2007.

[7] K. Bhattacharya et al. A model-driven approach to industrializing discovery processes in pharmaceutical research. *IBM Systems Journal*, 44(1), 2005.

[8] K. Bhattacharya, C. E. Gerede, R. Hull, R. Liu, and J. Su. Towards formal analysis of artifact-centric business process models. In *BPM*, 2007.

[9] BizAgi and Cordys and IBM and Oracle and SAP AG and Singularity (OMG Submitters) and Agile Enterprise Design and Stiftelsen SINTEF and TIBCO and Trisotech (Co-Authors). Case Management Model and Notation (CMMN), FTF Beta 1, Jan. 2013. OMG Document Number dtc/2013-01-01, Object Management Group.

[10] D. Boaz, L. Limonad, and M. Gupta. BizArtifact: Artifact-centric Business Process Management, June 2013. `http://sourceforge.net/projects/bizartifact/`.

[11] M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *LICS*, 2006.

[12] A. Bouajjani, P. Habermehl, Y. Jurski, and M. Sighireanu. Rewriting systems with data. In *FCT'07*.

[13] A. Bouajjani, P. Habermehl, and R. Mayr. Automatic verification of recursive procedures with one integer parameter. *Theoretical Computer Science*, 295:85–106, 2003.

[14] A. Bouajjani, Y. Jurski, and M. Sighireanu. A generic framework for reasoning about dynamic networks of infinite-state processes. In *TACAS'07*.

[15] P. Bouyer. A logical characterization of data languages. *Inf. Processing Letters*, 84(2), 2002.

[16] P. Bouyer, A. Petit, and D. Thérien. Algebraic approach to data languages and timed languages. *Inf. and Comp.*, 182(2), 2003.

[17] M. Brambilla, S. Ceri, S. Comai, P. Fraternali, and I. Manolescu. Specification and design of workflow-driven hypertexts. *Journal of Web Engineering*, 1(1), 2002.

[18] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification of infinite structures. In *Handbook of Process Algebra*, pages 545–623. Elsevier Science, 2001.

[19] D. Calvanese, G. De Giacomo, and M. Montali. Foundations of data-aware process analysis: a database theory perspective. In *PODS*, 2013.

[20] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing data-intensive Web applications.* Morgan-Kaufmann, 2002.

[21] T. Chao et al. Artifact-based transformation of IBM Global Financing: A case study. In *BPM*, 2009.

[22] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* MIT Press, 2000.

[23] D. Cohn, P. Dhoolia, F. Heath, F. Pinel, and J. Vergo. Siena: From powerpoint to web app in 5 minutes. In *ICSOC*, 2008.

[24] E. Damaggio, A. Deutsch, and V. Vianu. Artifact systems with data dependencies and arithmetic. In *ICDT*, 2011.

[25] E. Damaggio, R. Hull, and R. Vaculín. On the equivalence of incremental and fixpoint semantics for business artifacts with guard-stage-milestone lifecycles. *Information Systems*, 38:561–584, 2013.

[26] G. De Giacomo, R. D. Masellis, and R. Rosati. Verification of conjunctive artifact-centric services. *Int. J. Cooperative Inf. Syst.*, 21(2):111–140, 2012.

[27] H. de Man. Case management: Cordys approach. BP Trends (`www.bptrends.com`), 2009.

[28] S. Demri and R. Lazić. LTL with the Freeze Quantifier and Register Automata. In *LICS*, 2006.

[29] S. Demri, R. Lazić, and A. Sangnier. Model checking freeze LTL over one-counter automata. In *FoSSaCS*, 2008.

[30] A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *ICDT*, 2009.

[31] A. Deutsch, Y. Li, D. Lorant, and V. Vianu. Personal communication, 2014.

[32] A. Deutsch, M. Marcus, L. Sui, V. Vianu, and D. Zhou. A verifier for interactive, data-driven web applications. In *SIGMOD*, 2005.

[33] A. Deutsch, L. Sui, and V. Vianu. Specification and verification of data-driven web services. In *PODS*, 2004.

[34] A. Deutsch, L. Sui, and V. Vianu. Specification and verification of data-driven web applications. *JCSS*, 73(3):442–474, 2007.

[35] A. Deutsch, L. Sui, and V. Vianu. Specification and verification of data-driven web services. *JCSS*, 73(3):442–474, 2007.

[36] A. Deutsch, L. Sui, V. Vianu, and D. Zhou. A system for specification and verification of interactive, data-driven Web applications. In *SIGMOD*, 2006.

[37] A. Deutsch, L. Sui, V. Vianu, and D. Zhou. Verification of communicating data-driven Web services. In *PODS*, pages 90–99, 2006.

[38] G. Dong, R. Hull, B. Kumar, J. Su, and G. Zhou. A framework for optimizing distributed workflow executions. In *DBPL*, 1999.

[39] E. A. Emerson. Temporal and modal logic. In J. V. Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, pages 995–1072. North-Holland Pub. Co./MIT Press, 1990.

[40] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT*, 2003.

[41] M. F. Fernández, D. Florescu, A. Y. Levy, and D. Suciu. Declarative specification of web sites with Strudel. *VLDB Journal*, 9(1), 2000.

[42] D. Florescu, K. Yagoub, P. Valduriez, and V. Issarny. WEAVE: A data-intensive web site management system(software demonstration). In *EDBT*, 2000.

[43] C. E. Gerede, K. Bhattacharya, and J. Su. Static analysis of business artifact-centric operational models. In *SOCA*, 2007.

[44] C. E. Gerede and J. Su. Specification and verification of artifact behaviors in business process models. In *ICSOC*, 2007.

[45] R. Glushko and T. McGrath. *Document Engineering: Analyzing and Designing Documents for Business Informatics and Web Services.* MIT Press, Cmabridge, MA, 2005.

[46] B. B. Hariri, D. Calvanese, G. De Giacomo, A. Deutsch, and M. Montali. Verification of

relational data-centric dynamic systems with external services. In *PODS*, 2013.

[47] R. Hull, E. Damaggio, R. D. Masellis, F. Fournier, M. Gupta, F. H. III, S. Hobson, M. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, and R. Vaculín. Business artifacts with guard-stage-milestone lifecycles: Managing artifact interactions with conditions and events. In *ACM DEBS*, 2011.

[48] R. Hull, F. Llirbat, B. Kumar, G. Zhou, G. Dong, and J. Su. Optimization techniques for data-intensive decision flows. In *ICDE*, 2000.

[49] 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.

[50] R. Hull and J. Su. Tools for design of composite web services. In *SIGMOD*, 2004.

[51] R. Hull, J. Su, and R. Vaculín. Data management perspectives on business process management: tutorial overview. In *SIGMOD*, 2013.

[52] M. Jurdzinski and R. Lazić. Alternation-free modal mu-calculus for data trees. In *LICS*, 2007.

[53] S. Kumaran, P. Nandi, T. Heath, K. Bhaskaran, and R. Das. ADoc-oriented programming. In *Symp. on Applications and the Internet (SAINT)*, 2003.

[54] R. Lazić, T. Newcomb, J. Ouaknine, A. Roscoe, and J. Worrell. Nets with tokens which carry data. In *ICATPN'07*.

[55] L. Libkin. *Elements of Finite Model Theory.* Springer, 2004.

[56] R. Liu, K. Bhattacharya, and F. Y. Wu. Modeling business contexture and behavior using business artifacts. In *CAiSE*, 2007.

[57] M. Marin, R. Hull, and R. Vaculín. Data centric bpm and the emerging case management standard: A short survey. In *BPM Workshops*, 2012.

[58] D. Martin et al. OWL-S: Semantic markup for web services, W3C Member Submission, November 2003. `http://www.daml.org/services/`.

[59] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[60] G. Mecca, P. Merialdo, and P. Atzeni. Araneus in the era of XML. *IEEE Data Engineering Bulletin*, 22(3):19–26, 1999.

[61] S. Merz. Model checking: a tutorial overview. In *Modeling and verification of parallel processes*. Springer-Verlag New York, 2001.

[62] M. L. Minsky. *Computation: finite and infinite machines.* Prentice-Hall, 1967.

[63] P. Nandi and S. Kumaran. Adaptive business objects – a new component model for business integration. In *Proc. Intl. Conf. on Enterprise Information Systems*, 2005.

[64] F. Neven, T. Schwentick, and V. Vianu. Finite State Machines for Strings Over Infinite Alphabets. *ACM Transactions on Computational Logic*, 5(3):403–435, 2004.

[65] A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3), 2003.

[66] A. Pnueli. The temporal logic of programs. In *FOCS*, 1977.

[67] L. Segoufin and S. Torunczyk. Automata based verification over linearly ordered data domains. In *STACS*, 2011.

[68] A. Sistla and E. Clarke. The complexity of propositional linear temporal logic. *J. of the ACM*, 32:733–749, 1985.

[69] D. Solomakhin, M. Montali, S. Tessaris, and R. D. Masellis. Verification of artifact-centric systems: Decidability and modeling issues. In *ICSOC*, 2013.

[70] M. Spielmann. Verification of relational transducers for electronic commerce. *JCSS.*, 66(1):40–65, 2003.

[71] W. M. P. van der Aalst. *Process Mining.* Springer, 2011.

[72] J. Wang and A. Kumar. A framework for document-driven workflow systems. In *BPM*, 2005.

[73] W.-D. Zhu et al. Advanced Case Management with IBM Case Manager. Available at `http://www.redbooks.ibm.com/abstracts/sg247929.html?Open`.