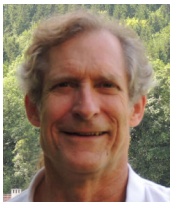
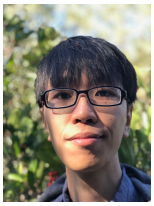
Automatic Verification of Database-Centric Systems

Alin Deutsch
UC San Diego



Richard Hull
IBM T.J. Watson Research Center



Yuliang Li
UC San Diego

Victor Vianu
UC San Diego & INRIA Paris

We present an overview of results on verification of temporal properties of infinite-state transition systems arising from processes that carry and manipulate unbounded data. The techniques bring into play tools from logic, database theory, and model checking. The theoretical results establish the boundaries of decidability and the complexity of verification for various models. We also describe verifier implementations with surprisingly good performance, suggesting that this line of research has real potential for practical impact.

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. This is a challenging task, because these are infinite-state systems due to the presence of unbounded data.

Classical software verification techniques such as model checking and theorem proving can be applied to such systems but 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.

More 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. Such tools 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 fully automatic verification. This relies on a marriage of techniques from logic, database theory and model checking.

In this article, we describe several models and results on automatic verification of database-driven systems, focusing on temporal properties of their underlying infinite-

state transition systems. To streamline the presentation, we focus on verification of business artifacts, and use it as a vehicle to introduce the main concepts 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 are a model of workflows in which data evolves under the action of "services" implementing business process tasks. They are a prominent exponent of data-aware business processes, that enrich the traditional process-centric approach by treating data as first-class citizens. The notion of business artifact was first introduced in [Nigam and Caswell 2003] and [Kumaran et al. 2003] (called there "adaptive documents"), and was further studied, from both practical and theoretical perspectives, in [Bhattacharya et al. 2007a; Gerede et al. 2007; Gerede and Su 2007; Bhattacharya et al. 2007b; Liu et al. 2007; Damaggio et al. 2013; Hull et al. 2011; Hariri et al. 2011]. Roots of the artifact model are present in "adaptive business objects" [Nandi and Kumaran 2005], "business entities", "document-driven" workflow [Wang and Kumar 2005] and "document" engineering [Glushko and McGrath 2005]. The Vortex framework [Hull et al. 1999; Dong et al. 1999; Hull et al. 2000] 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 [McIlraith et al. 2001; Martin et al. 2004] 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 [Cohn et al. 2008], a tool for compiling artifact-based procedural specifications into code supporting the corresponding business process. Its open-source descendant is the BizArtifact suite [Boaz et al. 2013]. More recently, the Guard-Stage-Milestone (GSM) approach [Damaggio et al. 2013; Hull et al. 2011] provides rich structuring mechanisms for services, including parallelism, concurrency and hierarchy, and has been incorporated in the OMG standard for Case Management Model and Notation (CMMN) [BizAgi and Cordys and IBM and Oracle and Singularity and SAP AG and Agile Enterprise Design and Stiftelsen SINTEF and TIBCO and Trisotech 2013; Marin et al. 2012].

2.1. Tuple Artifact Systems

We first consider a minimalistic variant of the artifact model, called *tuple artifact system* (TAS), then extend the results to a richer model capturing the core elements of the GSM artifact model. In a TAS, the artifact consists simply of a tuple of values that evolves throughout the workflow. We describe TAS informally, relying on an example (the formal development is provided in [Deutsch et al. 2009; Damaggio et al. 2012]). 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. The artifact is an evolving tuple of values, referred to by variables (sometimes called *attributes*). We use 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 is a finite first-order structure over a relational signature (called *schema* in database parlance), consisting of the following relations whose coordinates are given names, called *attributes*. Underlined attributes denote *keys*, which are attributes that uniquely identify each tuple in the relation.

PRODUCTS(id, price, availability, weight)
COUPONS(code, type, value, min_value, free_shiptype)
SHIPPING(type, cost, max_weight)
OFFERS(prod_id, discounted_price, active)

The database also satisfies the following inclusions:

COUPONS[free_shiptype] $\subseteq$ SHIPPING[type] and
OFFERS[prod_id] $\subseteq$ PRODUCTS[id].

The first inclusion says that each free_shiptype value in the COUPONS relation is also a type value in the SHIPPING relation. The second states that every prod_id value in the OFFERS is the actual id of a product in the PRODUCTS relation. In database terminology, it is said that free_shiptype and prod_id are *foreign keys*.

Services. Recall that artifacts evolve under the action of services. Each service is specified declaratively by a pre-condition π and a post-condition ψ, 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 ψ refers simultaneously to the current and *next* artifact values, as well as the database. In addition, both π and ψ may use arithmetic constraints on the variables, consisting of linear inequalities with integer coefficients.

Figure 1 shows some of the services for the business process of 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.

Notice the arithmetic computation used 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.

choose_product: The customer chooses a product.
π : $\texttt{status}$ = "edit_prod"
$\psi : \exists p, a, w(\texttt{PRODUCTS}(\texttt{prod\_id}', p, a, w) \wedge a > 0) \wedge \texttt{status}' =$ "edit_shiptype"
choose_shiptype: The customer chooses a shipping option.
π : $\texttt{status}$ = "edit_ship"
$\psi : \exists c, l, p, a, w(\texttt{SHIPPING}(\texttt{ship\_type}', c, l) \wedge \texttt{PRODUCTS}(\texttt{prod\_id}, p, a, w) \wedge l > w) \wedge$
$\texttt{status}' =$ "edit_coupon" $\wedge\ \texttt{prod\_id}' = \texttt{prod\_id}$
apply_coupon: The customer optionally inputs a coupon number.
π : $\texttt{status}$ = "edit_coupon"
$\psi : (\texttt{coupon}' = \lambda \wedge \exists p, a, w, c, l(\texttt{PRODUCTS}(\texttt{prod\_id}, p, a, w) \wedge$
$\texttt{SHIPPING}(\texttt{ship\_type}, c, l) \wedge \texttt{amount\_owed}' = p + c) \wedge \texttt{status}' =$ "processing"
$\wedge \texttt{prod\_id}' = \texttt{prod\_id} \wedge \texttt{ship\_type}' = \texttt{ship\_type}) \vee$
$(\exists t, v, m, s, p, a, w, c, l(\texttt{COUPONS}(\texttt{coupon}', t, v, m, s) \wedge$
$\texttt{PRODUCTS}(\texttt{prod\_id}, p, a, w) \wedge \texttt{SHIPPING}(\texttt{ship\_type}, c, l) \wedge p + c \geq m \wedge$
$(t =$ "free_shipping" $\rightarrow (s = \texttt{ship\_type} \wedge \texttt{amount\_owed}' = p)) \wedge$
$(t =$ "discount" $\rightarrow \texttt{amount\_owed}' = p + c - v))$
$\wedge \texttt{status}' =$ "processing" $\wedge\ \texttt{prod\_id}' = \texttt{prod\_id} \wedge \texttt{ship\_type}' = \texttt{ship\_type})$

Fig. 1. Three services

Semantics. The semantics of a TAS $\mathcal{A}$ consists of its *runs*. Given a database D, a run of $\mathcal{A}$ is an infinite sequence $\{\rho_i\}_{\geq 0}$ of artifact tuples such that ρ_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 ρ_i and D satisfy the pre-condition of S and ρ_i, ρ_{i+1} and D satisfy its post-condition. For uniformity, blocking prefixes of runs are extended to infinite runs by repeating forever their last tuple.

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.

Specifying Temporal Properties

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 3. 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 [Pnueli 1977]). 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 tuples 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 tuples. 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 $\mathbf{G}(p \rightarrow \mathbf{F}q)$, 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

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

Note that, as one would expect, the global variable x is universally quantified at the outermost scope. We say that an artifact system $\mathcal{A}$ satisfies an LTL-FO sentence φ if all runs of the artifact system satisfy φ 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 φ_2 for the running example, expressed by the LTL-FO formula

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

Property φ_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 [Emerson 1990; Spielmann 2003]. 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 [Demri and Lazić 2009; Demri et al. 2008].

Automatic Verification of Tuple 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

[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.

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. [Clarke et al. 2000; Merz 2000]).

Consider now a TAS $\mathcal{A}$ and an LTL-FO property φ. Model checking $\mathcal{A}$ with respect to φ can be viewed once again as a search for a counterexample run of $\mathcal{A}$, i.e. a run violating φ. 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 [Burkart et al. 2001] 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 [Bouajjani et al. 2003], rewriting systems with data [Bouajjani et al. 2007b; Bouajjani et al. 2007a], Petri nets with data associated to tokens [Lazić et al. 2008], automata and logics over infinite alphabets [Bouyer et al. 2003; Bouyer 2002; Neven et al. 2004; Demri and Lazić 2009; Jurdzinski and Lazic 2007; Bojanczyk et al. 2006; Bouajjani et al. 2007a], and temporal logics manipulating data [Demri and Lazić 2009; Demri et al. 2008]. However, the restricted use of data and the particular properties verified have limited applicability to the business artifact setting, or other database-driven applications.

Tuple artifacts without constraints or dependencies. We consider first tuple artifact systems and properties without arithmetic constraints or data dependencies. This case was studied in [Deutsch et al. 2009], 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 2.1. *It is decidable, given a TAS $\mathcal{A}$ with no data dependencies or arithmetic constraints, and an LTL-FO property φ with no arithmetic constraints, whether $\mathcal{A}$ satisfies φ.*

The complexity of verification is PSPACE-complete for fixed-arity database, 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 [Sistla and Clarke 1985].

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 tuples 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 [Deutsch et al. 2009] 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 [Segoufin and Toruńczyk 2011] using automata-theoretic techniques. In both cases, the worst-case complexity remains PSPACE.

Tuple artifacts with arithmetic constraints and data dependencies. Unfortunately, Theorem 2.1 fails even in the presence of simple data dependencies or arithmetic. Specifically, as shown in [Deutsch et al. 2009; Damaggio et al. 2012], 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 TAS 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 [Minsky 1967]. To simulate counter machines, a TAS uses an attribute for each counter. A service performs an increment (or decrement) operation 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 [Damaggio et al. 2012] 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 [Damaggio et al. 2012] 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 TAS and the property to be verified. We omit here the rather technical details. It is shown in [Damaggio et al. 2012] that feedback freedom of a TAS 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. 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 2.2. *[Damaggio et al. 2012] It is decidable, given a TAS $\mathcal{A}$ whose database satisfies a set of key and foreign key constraints, and an LTL-FO property φ such that $(\mathcal{A}, \varphi)$ is feedback free, whether every run of $\mathcal{A}$ on a valid database satisfies φ.*

The intuition behind decidability is the following. Recall the verification algorithm of Theorem 2.1. Because of the data dependencies and arithmetic constraints, the isomorphism types of symbolic runs no longer suffice, because every artifact tuple 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 tuples. 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 [Libkin 2004]). 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. However, the complexity of the resulting algorithm is non-elementary (a tower of exponentials of height k^2).

One might wonder if the decidability 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 above 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 [Deutsch et al. 2007].

2.2. Hierarchical Artifact Systems

While tuple artifact systems capture simple workflows, the model lacks many features present in full-fledged specification tools such as Guard-Stage-Milestone (GSM). We next describe briefly more recent work [Deutsch et al. 2016a; 2016b] that extends verification to a model called Hierarchical Artifact System (HAS), that captures the core of GSM. In particular, HAS features task hierarchy, concurrency, richer artifact data (including updatable artifact relations), and polynomial inequalities over the reals. We describe the HAS model informally by means of an example of a simple travel booking process inspired by Expedia, whereby customers create, store and retrieve candidate trips consisting of a hotel and/or flight reservation, and book their final choice of trip. The HAS uses a database with the following schema:

$$\texttt{FLIGHTS}(\texttt{ID}, \texttt{price}, \texttt{comp\_hotel\_id}) \quad \texttt{HOTELS}(\texttt{ID}, \texttt{unit\_price}, \texttt{discount\_price})$$

Intuitively, each flight stored in the FLIGHTS table has a hotel compatible for discount. If a flight is purchased together with a compatible hotel reservation, a discount is applied on the hotel reservation. Otherwise, the full price needs to be paid.

The process is carried out using a set of tasks forming a hierarchy (rooted tree). The travel booking artifact system has the following 4 tasks: T_1:**ManageTrips**, T_2:**AddHotel**, T_3:**AddFlight** and T_4:**BookTrip**, which form the hierarchy represented in Fig. 2.

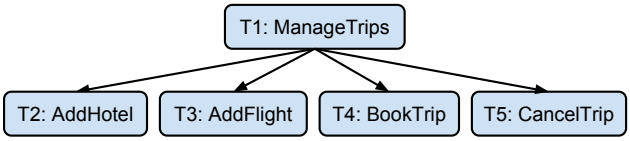


Fig. 2. Task hierarchy for the travel booking workflow

The workflow can be described informally as follows. At the root task **ManageTrips**, the customer can add a flight and/or hotel to the trip by calling the **AddHotel** or the **AddFlight** tasks. The customer can also store candidate trips in an artifact relation TRIPS and retrieve or delete previously stored trips, in analogy to Expedia's "My Itineraries" and "scratchpad" functionality. After the customer has made a decision, the **BookTrip** task is called to book the trip and the payment is processed. After the payment, the customer can decide to cancel the flight and/or the hotel reservation using the **CancelTrip** task and receive a refund.

Each task has associated to it local evolving data consisting of a tuple of artifact variables and an updatable artifact relation. For example, **ManageTrips** has the following artifact variables: flight_id, hotel_id, amount_paid, status. It also has an artifact relation TRIPS storing candidate trips (flight_id, hotel_id). An instance of the data held by this task, together with the underlying database, are shown in Figure 3.

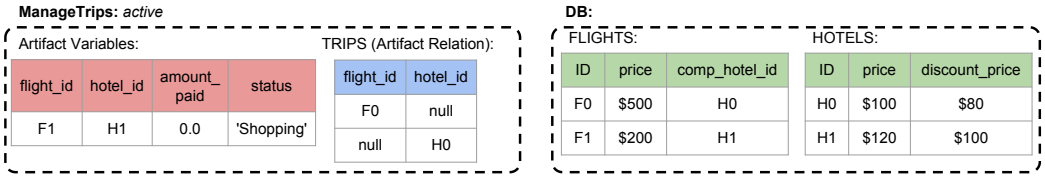


Fig. 3. Artifact data with underlying database

Each task also has an associated set of *services*. Each application of a service is guarded by an ∃FO pre-condition on the database and local data and causes an update of the local data, specified by an ∃FO post condition (constraining the next artifact tuple) and an insertion or retrieval of a tuple from the artifact relation. The conditions may include arithmetic constraints on the artifact variables in the form of polynomial inequalities with integer coefficients (interpreted over the reals). In addition, a task may invoke a child task with a tuple of parameters, and receive back a result if the child task completes. A run of the artifact system consists of an infinite sequence of transitions obtained by any valid interleaving of concurrently running task services.

Hierarchical LTL-FO

Properties of HAS are specified in a restriction of LTL-FO, called *hierarchical* LTL-FO (HLTL-FO). Intuitively, an HLTL-FO formula uses as building blocks LTL-FO formulas acting on runs of individual tasks, called local runs, referring only to the database and local data of the task, and can recursively state HLTL-FO properties on runs resulting from calls to children tasks. The language HLTL-FO closely fits the computational model and is also motivated on technical grounds. A main justification for adopting HLTL-FO is that LTL-FO (and even LTL) properties are undecidable for HAS's. This is shown by reduction from repeated state reachability in vector addition systems with states (VASS) with resets and bounded lossiness, whose undecidability follows from [Mayr 2003]. Another technical argument in favor of HLTL-FO is that it only expresses properties that are invariant under interleavings of independent tasks. Interleaving invariance is not only a natural soundness condition, but also allows more efficient model checking by *partial-order reduction* [Peled 1994]. Moreover, HLTL-FO enjoys a pleasing completeness property: it expresses, in a reasonable sense, *all* interleaving-invariant LTL-FO properties of HAS's. The proof, provided in [Deutsch et al. 2016b], is non-trivial, building on completeness results for propositional temporal logics on Mazurkiewicz traces [Diekert and Gastin 2004; 2006].

Verification of hierarchical artifact systems

Hierarchical artifact systems as sketched above provide powerful extensions to the variants previously studied, each of which immediately leads to undecidability of verification if not carefully controlled. [Deutsch et al. 2016a; 2016b] put forward a package of restrictions that ensures decidability while capturing a significant subset of the GSM model. This requires a delicate balancing act aiming to limit the dangerous features while retaining their most useful aspects. The intuition behind the restrictions is similar to feedback freedom, discussed prior to Theorem 2.2: they prevent dangerous data flow among artifact variables in the course of runs, that would enable recursive computations. However, in contrast to feedback freedom, this is automatically ensured by the restrictions without the need for additional syntactic checks. The restrictions are shown to be necessary by undecidability results.

The roadmap to verification is the following. Let Γ be a HAS and φ an HLTL-FO property for Γ. To verify that every run of Γ satisfies φ, we check that there is no run

satisfying $\neg\varphi$. As for TAS, there are infinitely many runs of Γ due so the unbounded data domain, so an exhaustive search is impossible. This problem is addressed by developing a symbolic representation of runs. The symbolic representation is subtle because, unlike the representations used to verify TAS in [Deutsch et al. 2009; Damaggio et al. 2012], it is no longer finite state. This is because summarizing the relevant information about artifact relations requires keeping track of the number of tuples of various isomorphism types. In addition, the symbolic representation must record arithmetic constraints satisfied by the artifact variables, which is done by partitioning the space into *cells* containing the points satisfying the same polynomial inequalities for all polynomials used in Γ and φ. This raises some subtle algorithmic issues that are addressed in [Deutsch et al. 2016a; 2016b]. Aside from arithmetic constraints, the symbolic representation keeps the following information for each active task T:

(1) an equality type of the artifact variables of T and the elements in the database reachable from them by navigating foreign keys up to a specified depth $h(T)$. This is called the T-*isomorphism type* of the variables.
(2) for non-root tasks, the T-isomorphism type of the input variables (passed when the task was called by its parent)
(3) for each T-isomorphism type of the set variables of T together with the input variables, the net number of insertions of tuples of that type in its artifact relation.

Intuitively, (1) and (2) are needed in order to ensure that the assumptions made about the database in the course of symbolic computations while navigating via foreign keys are consistent. The depth $h(T)$ is chosen to be sufficiently large to ensure consistency. (3) is required in order to make sure that a retrieval from the artifact relation of a tuple with a given T-isomorphism type is allowed only when sufficiently many tuples of that type have been previously inserted. Because of (3), the symbolic representation can be viewed as a Vector Addition System with States (VASS) [Blockelet and Schmitz 2011]. The verification algorithm relies on reductions to state reachability problems in the corresponding VASS. The worst-case complexity is non-elementary, as for feedback-free TAS [Damaggio et al. 2012]. However, the height of the tower of exponentials in [Damaggio et al. 2012] is the square of the total number of artifact variables of the system, whereas for HAS it is the depth of the hierarchy, likely to be much smaller.

Finally, the complexity is considerably lower in special cases that occur frequently in practice, such as when the schema is *acyclic* (meaning that the graph of foreign key references among database relations is acyclic). This property is shared by the common Star (or Snowflake) schemas [Kimball and Ross 2011; Vassiliadis and Sellis 1999]. Section 4.3 describes the implementation of a verifier based on these results.

2.3. Other work on verification of artifact systems

Initial work on formal analysis of artifact-based business processes in restricted contexts has investigated reachability [Gerede et al. 2007; Gerede and Su 2007], general temporal constraints [Gerede and Su 2007], and the existence of complete execution or dead end [Bhattacharya et al. 2007b]. For each considered problem, verification is generally undecidable; decidability results were obtained only under rather severe restrictions, e.g., restricting all pre-conditions to be "true" [Gerede et al. 2007], restricting to bounded domains [Gerede and Su 2007; Bhattacharya et al. 2007b], or restricting the pre- and post-conditions to be propositional, and thus not referring to data values [Gerede and Su 2007]. [Calvanese et al. 2009] adopts an artifact model variation with arithmetic operations but no database. Decidability relies on restricting runs to bounded length. [Zhao et al. 2009] addresses the problem of the existence of a run that satisfies a temporal property, for a restricted case with no database and only propo-

sitional LTL properties. None of these works model an underlying database, artifact relations, task hierarchy, or arithmetic.

A more recent line of work has tackled the verification of artifact systems in which properties are checked only over the runs starting from a given initial database that may evolve via updates, insertions and deletions. [Belardinelli et al. 2011b; 2011a; 2012a; 2012b; De Giacomo et al. 2012] consider several models and property languages, culminating in [Hariri et al. 2013], which addresses verification of first-order μ-calculus (hence branching time) properties in a framework that is equivalent to artifact systems whose input is provided by external services. [Belardinelli et al. 2014; Calvanese et al. 2015] extend the results of [Hariri et al. 2013] to artifact-centric multi-agent systems where the property language is a version of first-order branching-time temporal-epistemic logic expressing the knowledge of the agents. This line of work uses variations of a business process model called DCDS (data-centric dynamic systems), which is sufficienty expressive to capture the GSM model, as shown in [Solomakhin et al. 2013]. In their unrestricted form, DCDS and HAS have similar expressive power. However, verification for DCDS is only considered for a *fixed* rather than arbitrary initial database. Recently, [Abdulla et al. 2016] considered verification of monadic second-order properties of runs in a model where the underlying database can be updated by insertions and deletions. Decidability is obtained subject to a restriction called k-recency boundedness, allowing only the most recent k elements in the database to be modified by an update, for a fixed k.

See [Hull et al. 2013] for a survey on data-centric business process management, and [Calvanese et al. 2013] for a survey of corresponding verification results.

3. 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 [Hull and Su 2004] for a tutorial.

A commercially successful high-level specification tool for web applications is Web Ratio [web 2018], an outgrowth of the earlier academic prototype WebML [Ceri et al. 2002; Brambilla et al. 2002]. We illustrate with an example the WebML approach to specifying data-driven web services, formally studied in [Deutsch et al. 2004; 2007]. Consider the common scenario of a web service 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. Figure 4 illustrates 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.

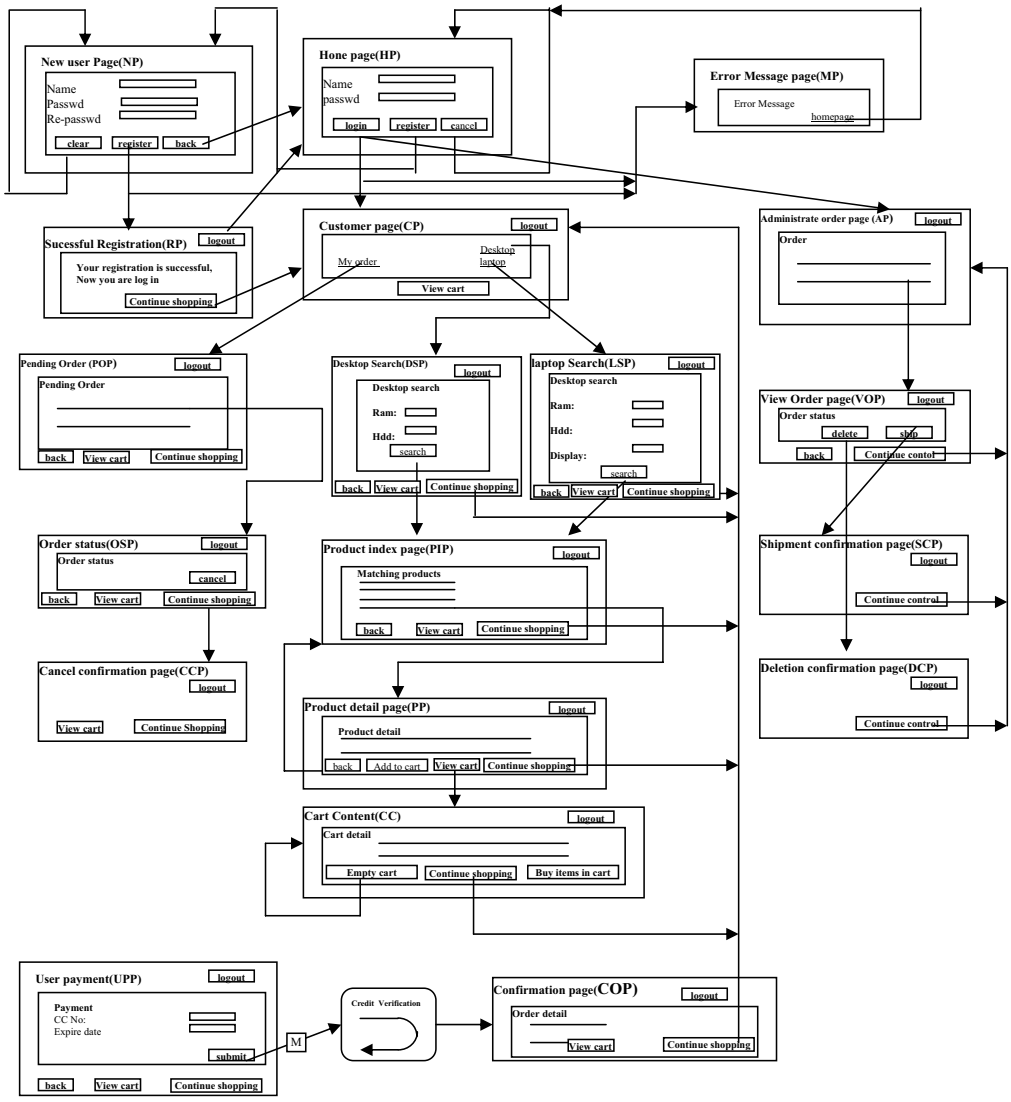


Fig. 4. Web pages in the computer shopping site.

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 model is an extension of the Abstract State Machine (ASM) transducer previously studied by Spielmann [Spielmann 2003]. Similarly to the earlier Relational Transducer [Abiteboul et al. 2000], the ASM$^+$ transducer models 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 motiva-

tion 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 [Spielmann 2003] 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 even small relaxations lead to undecidability. Second, as argued in [Deutsch et al. 2004; 2007], 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 4 can be modeled under this restriction, and many relevant natural properties can be expressed. Third, as in the case of tuple 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 in Section 4.

Compositions of ASM$^+$ Transducers. The verification results discussed above apply to single ASM$^+$ transducers in isolation. These results were extended in [Deutsch et al. 2006b] to the more challenging 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, the input-boundedness restriction is extended in the natural way. 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.

As in the case of single transducers, verification becomes undecidable if some of the restrictions are relaxed. Not suprisingly, verification is undecidable with unbounded queues (this already happens for finite-state systems [Brand and Zafiropulo 1983]). More interestingly, lossiness of channels is essential: verification becomes undecidable under the assumption that channels are *perfect*, i.e. messages are never lost (the proof is by reduction of the Post Correspondence Problem [Post 1947]).

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 [Deutsch et al. 2006b] for modular verification, subject to an appropriate extension of the input-boundedness restriction.

4. IMPLEMENTATION

The high worst-case complexity of the verification algorithms discussed earlier raises the question of whether automatic verification is practically feasible at all. We discuss next several implementations of verifiers for data-centric systems, showing that significant classes of systems and properties can indeed be efficiently verified.

4.1. The WAVE Verifier

While the PSPACE upper bound obtained for verification of ASM^+ transducers 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 [Deutsch et al. 2006b; Deutsch et al. 2005].

The verifier, as well as its target specification framework, are both implemented from scratch. First, a tool is developed for high-level, efficient specification of data-driven Web services, in the spirit of WebML. Next, WAVE is implemented 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. 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 symbolic 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 show very 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 [Deutsch et al. 2005], and a demo of the WAVE prototype was presented in [Deutsch et al. 2006a].

4.2. The SpinArt verifier

Recall that, simlarly to input-blunded ASM transducers, the worst-case complexity of the verification algorithm for tuple artifact systems (TAS) is PSPACE. [Li et al. 2017a] explore the possibility of using the off-the-shelf model checker Spin to verify a variant of TAS, with the ultimate goal of automatically verifying HAS specifications. The model is expressive enough to allow data of unbounded domain and size, which are not directly supported by Spin or other state-of-the-art model checkers. Therefore, a direct translation into Spin requires setting limits on the size of the data and its domain, resulting in an incomplete verifier. To address this challenge, the symbolic verification techniques establishing the decidability results in [Deutsch et al. 2016a] are used to develop a simple algorithm for translating TAS specifications and properties into equivalent problem instances that can be verified by Spin, without sacrificing either the soundness or the completeness of the verifier. However, a naive use of Spin still results in poor performance even with the translation algorithm. Therefore, an array of nontrivial optimizations techniques were developed to render verification tractable. SpinArt is the first implementation of an artifact system verifier that preserves decidability under unbounded data while being based on off-the-shelf model checking technology.

The main contributions of [Li et al. 2017a] are the following. By exploiting the symbolic verification approach from previous work [Deutsch et al. 2016a; Damaggio et al. 2012], a simple algorithm is exhibited for translating the verification problem into an equivalent instance in Spin. This algorithm forms the basis of the implementation of SpinArt. In addition, SpinArt uses two nontrivial optimization techniques to achieve satisfactory performance. The first consists of a more efficient translation algorithm avoiding a quadratic blowup in the size of the specification due to keys and foreign keys, so that it shortens significantly the compilation and execution time for Spin. The second optimization is based on static analysis, and greatly reduces the size of the search space by exploiting constraints extracted from the input specification during a pre-computation phase. Although these techniques are designed with Spin as the target tool, they can likely be adapted to implementations based on other off-the-shelf model checkers.

The performance of SpinArt is evaluated experimentally using both real-world and synthetic data-driven workflows and properties. A benchmark of real and synthetic artifact systems and LTL-FO properties was created from existing sets of business process specifications and temporal properties by extending them with data-aware features. The experiments highlight the impact of the optimizations and various parameters of the specifications and properties on the performance of SpinArt. Over 384 runs on real workflows (32 workflows rewritten from real BPMN workflows with 12 LTL-FO properties on each workflow), SpinArt achieves an average running time of 2.97 seconds with 3 failed runs. The results also show that the two optimizations significantly reduce the size of the state space and the compilation time. However, SpinArt does not scale well on the synthetic workflows. Among 1440 runs on synthetic workflows, the average running is 83.98s, with $>30\%$ failures due to memory overflow or timeout.

4.3. The VERIFAS verifier

As discussed above, the SpinArt verifier cannot handle some of the most useful features of the HAS model, such as artifact relations holding an unbounded number of tuples. Moreover, its performance is somewhat disappointing even after deploying a battery of non-trivial optimizations. This suggests limited usefulness of off-the-shelf tools in artifact verification and the need for tailored approaches. [Li et al. 2017b; Deutsch et al. 2017] present VERIFAS, an artifact verifier implemented from scratch. VERIFAS applies to a variant of HAS which strikes a practically relevant trade-off between expressivity and verification complexity. Its modeling capabilities are demonstrated in [Li et al. 2017b; Deutsch et al. 2017] by its ability to specify a realistic benchmark of business processes. While the version of VERIFAS currently implemented does not yet handle arithmetic, the core verification algorithm can be augmented to include arithmetic along the lines developed for HAS in [Deutsch et al. 2016a; 2016b].

The implementation makes crucial use of novel optimization techniques, with significant impact on performance. The optimizations are non-trivial and include concise symbolic representations, aggressive pruning in the search algorithm, and the use of highly efficient data structures. We briefly discuss the main ideas. The starting point for the implementation is the verification algorithm outlined in [Deutsch et al. 2016a; 2016b], which reduces verification to repeated state reachability problems in a VASS constructed form the HAS and the property to be verified. However, implementation of an efficient verifier is challenging. The algorithm that directly translates the artifact specification and the property into a VASS and checks (repeated) reachability is impractical because the resulting VASS can have exponentially many states and counters in the input size, and state-of-the-art VASS tools can only handle a small number of counters (<100) [mis 2017]. To mitigate the inefficiency, VERIFAS never generates the whole VASS but instead lazily generates only *reachable* symbolic states, whose

number is usually much smaller. In addition, isomorphism types in the symbolic representation are replaced by *partial isomorphism types*, which store only the subset of constraints on the variables imposed by the current run, leaving the rest unspecified. This representation is not only more compact, but also results in a significantly smaller search space in practice. The implementation of the (repeated) state reachability test is based on a series of optimizations to the classic Karp-Miller algorithm [Karp et al. 1972], based on a novel, more aggressive pruning of the search space.

As for SpinArt, the performance of VERIFAS is evaluated using both real-world and synthetic artifact systems and properties from a benchmark created by bootstrapping from existing sets of business process specifications and properties by extending them with data-aware features. The experiments highlight the impact of the various optimizations and parameters of both the artifact systems and properties. The performance of VERIFAS significantly improves over SpinArt. On the real set of workflows, even when the tasks have updatable artifact relations, VERIFAS achieves an $>$10x improvement in the average running time (0.245 seconds) with no failed runs. Also on the synthetic set of workflows, it is able to scale with an average running time of 11.01 second and fail due to timeout on $<$1% of the runs. In addition, the experiments show that the verification time of VERIFAS increases exponentially with the input workflow's cyclomatic complexity, a classic metric for measuring the complexity of program modules [Watson et al. 1996]. The performance is within a reasonable range when the cyclomatic complexity is below the maximal value recommended by software engineering practice. Thus, VERIFAS performs very well on practically relevant classes of artifact systems. Compared to the Spin-based verifier, it not only applies to a much broader class of artifacts but also has a decisive performance advantage even on the simple artifacts the Spin-based verifier is able to handle.

5. CONCLUSIONS

Database-centric 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 tools specifying such systems by logical rules that are automatically compiled into code. These high-level specifications provide a natural target for verification.

In contrast to general-purpose software verification that loses semantics by abstracting away data, or provides semi-automatic and incomplete solutions based on theorem proving, we focus on fully-automatic sound and complete verification. In this setting, verification amounts to deciding the unsatisfiability of temporal logic statements (corresponding to negated correctness properties) over infinite-state transition systems specified by logical rules. The results we described suggest that such verification is feasible for significant classes of database-centric systems specified by high-level tools.

We focused on two representative models, business artifacts and data-driven web services, and identified restrictions guaranteeing decidability of verification. The described algorithms involve symbolic model checking, that reduces verification to finite-state model checking or to state reachability problems in Vector Addition Systems with States (equivalently, Petri nets). The worst-case complexity of the algorithms ranges from PSPACE to non-elementary.

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 [Deutsch et al. 2005], SpinArt [Li et al. 2017a] and VERIFAS [Li et al. 2017b; Deutsch et al. 2017] verifiers is therefore particularly encouraging. Like the theoretical results, this is made possible by a novel coupling of logic, database and model checking techniques.

There are several promising future work opportunities towards improving the applicability and impact of the presented results.

On the theoretical front, the quest for the right package of restrictions that enables verification while capturing more relevant sets of specifications is still ongoing. For example, real-life artifacts often require the ability to aggregate data collections (e.g. summing up all items in the shopping cart). Other useful extensions involve checking properties of the interaction among multiple actors in the database-centered system, or among multiple artifact instances evolving in parallel. The inter-operation, evolution, and integration of multiple systems 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 [Solomakhin et al. 2013]).

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.

We believe that the research described in this atticle may be just the starting point of a fruitful marriage between the database and computer-aided verification areas.

REFERENCES

2017. MIST - a safety checker for Petri Nets and extensions. https://github.com/pierreganty/mist/wiki. (2017).

2018. Web Ratio. (2018). http://www.webratio.com/.

Parosh Aziz Abdulla, C. Aiswarya, Mohamed Faouzi Atig, Marco Montali, and Othmane Rezine. 2016. Recency-Bounded Verification of Dynamic Database-Driven Systems. In *PODS*. 195–210.

Serge Abiteboul, Victor Vianu, Brad Fordham, and Yelena Yesha. 2000. Relational transducers for electronic commerce. *JCSS* 61, 2 (2000), 236–269.

Francesco Belardinelli, Alessio Lomuscio, and Fabio Patrizi. 2011a. A Computationally-Grounded Semantics for Artifact-Centric Systems and Abstraction Results. In *IJCAI*. 738–743.

Francesco Belardinelli, Alessio Lomuscio, and Fabio Patrizi. 2011b. Verification of Deployed Artifact Systems via Data Abstraction. In *ICSOC*.

Francesco Belardinelli, Alessio Lomuscio, and Fabio Patrizi. 2012a. An Abstraction Technique for the Verification of Artifact-Centric Systems. In *KR*.

Francesco Belardinelli, Alessio Lomuscio, and Fabio Patrizi. 2012b. Verification of GSM-Based Artifact-Centric Systems through Finite Abstraction. In *ICSOC*.

Francesco Belardinelli, Alessio Lomuscio, and Fabio Patrizi. 2014. Verification of Agent-Based Artifact Systems. *J. Artif. Intell. Res. (JAIR)* 51 (2014), 333–376.

Kamal Bhattacharya, Nathan S Caswell, Santhosh Kumaran, Anil Nigam, and Frederick Y Wu. 2007a. Artifact-centered operational modeling: Lessons from customer engagements. *IBM Systems Journal* 46, 4 (2007), 703–721.

Kamal Bhattacharya, Cagdas Gerede, Richard Hull, Rong Liu, and Jianwen Su. 2007b. Towards formal analysis of artifact-centric business process models. In *International Conference on Business Process Management*. Springer, 288–304.

BizAgi and Cordys and IBM and Oracle and Singularity and SAP AG and Agile Enterprise Design and Stiftelsen SINTEF and TIBCO and Trisotech. 2013. Case Management Model and Notation (CMMN), FTF Beta 1. (Jan. 2013). http://www.omg.org/spec/CMMN/1.0/Beta1/ OMG Document Number dtc/2013-01-01, Object Management Group.

Michel Blockelet and Sylvain Schmitz. 2011. Model checking coverability graphs of vector addition systems. In *Mathematical Foundations of Computer Science 2011*. Springer, 108–119.

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

Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. 2006. Two-variable logic on words with data. In *LICS*. IEEE, 7–16.

Ahmed Bouajjani, Peter Habermehl, Yan Jurski, and Mihaela Sighireanu. 2007a. Rewriting systems with data. In *International Symposium on Fundamentals of Computation Theory*. Springer, 1–22.

Ahmed Bouajjani, Peter Habermehl, and Richard Mayr. 2003. Automatic verification of recursive procedures with one integer parameter. *Theoretical Computer Science* 295, 1-3 (2003), 85–106.

Ahmed Bouajjani, Yan Jurski, and Mihaela Sighireanu. 2007b. A generic framework for reasoning about dynamic networks of infinite-state processes. In *TACAS*. Springer, 690–705.

Patricia Bouyer. 2002. A logical characterization of data languages. *Inform. Process. Lett.* 84, 2 (2002), 75–85.

Patricia Bouyer, Antoine Petit, and Denis Thérien. 2003. An algebraic approach to data languages and timed languages. *Information and Computation* 182, 2 (2003), 137–162.

Marco Brambilla, Stefano Ceri, Sara Comai, Piero Fraternali, and Ioana Manolescu. 2002. Specification and design of workflow-driven hypertexts. *Journal of Web Engineering* 1, 2 (2002), 163–182.

Daniel Brand and Pitro Zafiropulo. 1983. On communicating finite-state machines. *J. of the ACM* 30, 2 (1983), 323–342.

Olaf Burkart, Didier Caucal, Faron Moller, and Bernhard Steffen. 2001. Verification on infinite structures. In *Handbook of Process algebra*. Elsevier, 545–623.

Diego Calvanese, Giuseppe De Giacomo, Richard Hull, and Jianwen Su. 2009. Artifact-centric workflow dominance. In *Service-Oriented Computing*. Springer, 130–143.

Diego Calvanese, Giuseppe De Giacomo, and Marco Montali. 2013. Foundations of data-aware process analysis: a database theory perspective. In *PODS*.

Diego Calvanese, Giorgio Delzanno, and Marco Montali. 2015. Verification of Relational Multiagent Systems with Data Types. In *AAAI*. 2031–2037.

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

Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. 2000. *Model Checking*. MIT Press.

David Cohn, Pankaj Dhoolia, Fenno Heath, Florian Pinel, and John Vergo. 2008. Siena: From PowerPoint to Web App in 5 Minutes. In *ICSOC*.

Elio Damaggio, Alin Deutsch, and Victor Vianu. 2012. Artifact systems with data dependencies and arithmetic. *ACM Transactions on Database Systems (TODS)* 37, 3 (2012), 22. Also in ICDT 2011.

Elio Damaggio, Richard Hull, and Roman VaculíN. 2013. On the equivalence of incremental and fixpoint semantics for business artifacts with Guard–Stage–Milestone lifecycles. *Information Systems* 38, 4 (2013), 561–584.

Giuseppe De Giacomo, Riccardo De Masellis, and Riccardo Rosati. 2012. Verification of Conjunctive Artifact-Centric Services. *Int. J. Cooperative Inf. Syst.* 21, 2 (2012), 111–140.

Stéphane Demri and Ranko Lazić. 2009. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic (TOCL)* 10, 3 (2009), 16.

Stéphane Demri, Ranko Lazić, and Arnaud Sangnier. 2008. Model checking freeze LTL over one-counter automata. In *International Conference on Foundations of Software Science and Computational Structures*. Springer, 490–504.

Alin Deutsch, Richard Hull, Fabio Patrizi, and Victor Vianu. 2009. Automatic verification of data-centric business processes. In *ICDT*. ACM, 252–267.

Alin Deutsch, Yuliang Li, and Victor Vianu. 2016a. Verification of hierarchical artifact systems. In *PODS*. 179–194.

Alin Deutsch, Yuliang Li, and Victor Vianu. 2016b. Verification of Hierarchical Artifact Systems. *arXiv preprint* arXiv:1604.00967v1 (2016).

Alin Deutsch, Yuliang Li, and Victor Vianu. 2017. VERIFAS: A Practical Verifier for Artifact Systems (Extended Version). *arXiv preprint* arXiv:1705.10007 (2017).

Alin Deutsch, Monica Marcus, Liying Sui, Victor Vianu, and Dayou Zhou. 2005. A verifier for interactive, data-driven web applications. In *SIGMOD*. ACM, 539–550.

Alin Deutsch, Liying Sui, and Victor Vianu. 2004. Specification and verification of data-driven web services. In *PODS*. ACM, 71–82.

Alin Deutsch, Liying Sui, and Victor Vianu. 2007. Specification and Verification of Data-driven Web services. *JCSS* 73, 3 (2007), 442–474.

Alin Deutsch, Liying Sui, Victor Vianu, and Dayou Zhou. 2006a. A system for specification and verification of interactive, data-driven web applications. In *SIGMOD*. ACM, 772–774.

Alin Deutsch, Liying Sui, Victor Vianu, and Dayou Zhou. 2006b. Verification of communicating data-driven web services. In *PODS*. ACM, 90–99.

Volker Diekert and Paul Gastin. 2004. Pure Future Local Temporal Logics Are Expressively Complete for Mazurkiewicz Traces. In *LATIN 2004: Theoretical Informatics, 6th Latin American Symposium, Buenos Aires, Argentina, April 5-8, 2004, Proceedings*. 232–241.

Volker Diekert and Paul Gastin. 2006. Pure future local temporal logics are expressively complete for Mazurkiewicz traces. *Inf. Comput.* 204, 11 (2006), 1597–1619.

Guozhu Dong, Richard Hull, Bharat Kumar, Jianwen Su, and Gang Zhou. 1999. A framework for optimizing distributed workflow executions. In *DBPL*. Springer, 152–167.

E. Allen Emerson. 1990. Temporal and Modal Logic. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, J. Van Leeuwen (Ed.). North-Holland Pub. Co./MIT Press, 995–1072.

Cagdas E Gerede, Kamal Bhattacharya, and Jianwen Su. 2007. Static analysis of business artifact-centric operational models. In *SOCA*. IEEE, 133–140.

Cagdas E Gerede and Jianwen Su. 2007. Specification and verification of artifact behaviors in business process models. In *ICSOC*. Springer, 181–192.

Robert J Glushko and Tim McGrath. 2005. *Document Engineering: Analyzing and Designing Documents for Business Informatics and Web Services MIT Press*.

Babak Bagheri Hariri, Diego Calvanese, Giuseppe De Giacomo, Riccardo De Masellis, and Paolo Felli. 2011. Foundations of relational artifacts verification. In *BPM*. Springer, 379–395.

Babak Bagheri Hariri, Diego Calvanese, Giuseppe De Giacomo, Alin Deutsch, and Marco Montali. 2013. Verification of relational data-centric dynamic systems with external services. In *PODS*.

Richard Hull, Elio Damaggio, Riccardo De Masellis, Fabiana Fournier, Manmohan Gupta, Fenno Terry Heath III, Stacy Hobson, Mark Linehan, Sridhar Maradugu, Anil Nigam, and others. 2011. Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In *DEBS*. ACM, 51–62.

Richard Hull, Francois Llirbat, Bharat Kumar, Gang Zhou, Guozhu Dong, and Jianwen Su. 2000. Optimization Techniques for Data Intensive Decision Flows. In *ICDE*.

Richard Hull, Francois Llirbat, Eric Siman, Jianwen Su, Guozhu Dong, Bharat Kumar, and Gang Zhou. 1999. Declarative workflows that support easy modification and dynamic browsing. *ACM SIGSOFT Software Engineering Notes* 24, 2 (1999), 69–78.

Richard Hull and Jianwen Su. 2004. Tools for design of composite web services. In *SIGMOD*. ACM, 958–961.

Richard Hull, Jianwen Su, and Roman Vaculín. 2013. Data management perspectives on business process management: tutorial overview. In *SIGMOD*.

Marcin Jurdzinski and Ranko Lazic. 2007. Alternation-free modal mu-calculus for data trees. In *LICS*. IEEE, 131–140.

Richard M Karp, Raymond E Miller, and Arnold L Rosenberg. 1972. Rapid identification of repeated patterns in strings, trees and arrays. In *Proc. ACM Symposium on Theory of Computing (STOC)*. ACM, 125–136.

Ralph Kimball and Margy Ross. 2011. The data warehouse toolkit: the complete guide to dimensional modeling. (2011).

Santhosh Kumaran, Prabir Nandi, Terry Heath, Kumar Bhaskaran, and Raja Das. 2003. ADoc-oriented programming. In *Symposium on Applications and the Internet*. IEEE, 334–341.

Ranko Lazić, Tom Newcomb, Joël Ouaknine, Andrew W Roscoe, and James Worrell. 2008. Nets with tokens which carry data. *Fundamenta Informaticae* 88, 3 (2008), 251–274.

Yuliang Li, Alin Deutsch, and Victor Vianu. 2017a. A Spin-based Verifier for Artifact Systems. *arXiv preprint* arXiv:1705.09427 (2017). http://arxiv.org/abs/1705.09427

Yuliang Li, Alin Deutsch, and Victor Vianu. 2017b. VERIFAS: A Practical Verifier for Artifact Systems. *PVLDB* 11, 3 (2017), 283–296.

Leonid Libkin. 2004. *Elements of Finite Model Theory*. Springer.

Rong Liu, Kamal Bhattacharya, and Frederick Y Wu. 2007. Modeling business contexture and behavior using business artifacts. In *International Conference on Advanced Information Systems Engineering*. Springer, 324–339.

Mike Marin, Richard Hull, and Roman Vaculín. 2012. Data Centric BPM and the Emerging Case Management Standard: A Short Survey. In *BPM Workshops*.

David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, and others. 2004. OWL-S: Semantic markup for web services. (2004).

Richard Mayr. 2003. Undecidable problems in unreliable computations. *Theoretical Computer Science* 297, 1 (2003), 337–354.

Sheila A McIlraith, Tran Cao Son, and Honglei Zeng. 2001. Semantic web services. *IEEE intelligent systems* 16, 2 (2001), 46–53.

Stephan Merz. 2000. Model checking: A tutorial overview. In *Summer School on Modeling and Verification of Parallel Processes*. Springer, 3–38.

Marvin L. Minsky. 1967. *Computation: finite and infinite machines*. Prentice-Hall.

Prabir Nandi and Santhosh Kumaran. 2005. Adaptive Business Objects-A new Component Model for Business Integration.. In *ICEIS (3)*. 179–188.

Frank Neven, Thomas Schwentick, and Victor Vianu. 2004. Finite state machines for strings over infinite alphabets. *ACM Transactions on Computational Logic (TOCL)* 5, 3 (2004), 403–435.

Anil Nigam and Nathan S Caswell. 2003. Business artifacts: An approach to operational specification. *IBM Systems Journal* 42, 3 (2003), 428–445.

Doron Peled. 1994. Combining partial order reductions with on-the-fly model-checking. In *Computer aided verification*. Springer, 377–390.

Amir Pnueli. 1977. The Temporal Logic of Programs. In *FOCS*.

E. L. Post. 1947. Recursive Unsolvability of a Problem of Thue. *J. of Symbolic Logic* 12 (1947), 1–11.

Luc Segoufin and Szymon Toruńczyk. 2011. Automata based verification over linearly ordered data domains. In *STACS*, Vol. 9. 81–92.

A Prasad Sistla and Edmund M Clarke. 1985. The complexity of propositional linear temporal logics. *JACM* 32, 3 (1985), 733–749.

Dmitry Solomakhin, Marco Montali, Sergio Tessaris, and Riccardo De Masellis. 2013. Verification of Artifact-Centric Systems: Decidability and Modeling Issues. In *ICSOC*. 252–266.

Marc Spielmann. 2003. Verification of relational transducers for electronic commerce. *JCSS* 66, 1 (2003), 40–65.

Panos Vassiliadis and Timos Sellis. 1999. A survey of logical models for OLAP databases. *ACM Sigmod Record* 28, 4 (1999), 64–69.

Jianrui Wang and Akhil Kumar. 2005. A framework for document-driven workflow systems. In *BPM*. Springer, 285–301.

Arthur Henry Watson, Dolores R Wallace, and Thomas J McCabe. 1996. *Structured testing: A testing methodology using the cyclomatic complexity metric*. Vol. 500. US Department of Commerce, Technology Administration, National Institute of Standards and Technology.

Xiangpeng Zhao, Jianwen Su, Hongli Yang, and Zongyan Qiu. 2009. Enforcing constraints on life cycles of business artifacts. In *Theoretical Aspects of Software Engineering*. IEEE, 111–118.