Automatic Verification of Data-Centric Business Processes*

Elio Damaggio[1], Alin Deutsch[1], Richard Hull[2,**], and Victor Vianu[1,***]

[1] University of California, San Diego
{elio,deutsch,vianu}@cs.ucsd.edu
[2] IBM T.J. Watson Research Center
hull@us.ibm.com

1 Introduction

Recent years have witnessed the evolution of business process specification frameworks from the traditional process-centric approach towards data-awareness. Process-centric formalisms focus on control flow while under-specifying the underlying data and its manipulations by the process tasks, often abstracting them away completely. In contrast, data-aware formalisms treat data as first-class citizens. A notable exponent of this class is the *business artifact model* pioneered in [46, 34], deployed by IBM in professional services offerings with associated tooling, and further studied in a line of follow-up works [4, 5, 26, 27, 6, 38, 33, 35, 42]. 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, yielding proven savings when performing business process transformations [5, 4, 15].

In recent work [21, 16], we addressed the verification problem for artifact systems, which consists in statically checking whether all runs of an artifact system satisfy desirable properties expressed in an extension of linear-time temporal logic (LTL). We consider a simplified model of artifact systems in which each artifact consists of a record of variables and the services may consult (though not update) an underlying database. Services can also set the artifact variables to new values from an infinite domain, thus modeling external inputs and tasks specified only partially, using pre- and post-conditions. Post-conditions permit non-determinism in the outcome of a service, to capture for instance tasks in which a human makes a final determination about the value of an artifact variable, subject to certain constraints. This setting results in a challenging infinite-state verification problem, due to the infinite data domain. Rather than relying on general-purpose software verification tools suffering from well-known limitations, the work of [21] and [16] addresses this problem by identifying relevant classes

* This work was supported by the National Science Foundation under award III-0916515, and by IBM Research through an Open Collaborative Research grant in conjunction with Project ArtiFact.

** This author was supported in part by the National Science Foundation under award IIS-0812578.

*** This author was supported in part by the European Research Council under grant Webdam, agreement 226513.

S. Rinderle-Ma, F. Toumani, and K. Wolf (Eds.): BPM 2011, LNCS 6896, pp. 3–16, 2011.

of artifact systems and properties for which fully automatic verification is possible. This extended abstract informally summarizes these results.

Context and Related Work

Data-aware business process models. The specific notion of business artifact was first introduced in [46] and [34] (called there "adaptive documents"), and was further studied, from both practical and theoretical perspectives, in [4, 5, 26, 27, 6, 38, 33, 35, 42, 25, 17, 29, 3]. (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" [43], "business entities", "document-driven" workflow [50] and "document" engineering [28]. The Vortex framework [31, 24, 30] also allows the specification of database manipulations and provides declarative specifications for when services are applicable to a given artifact.

The artifact model considered here is inspired in part by the field of semantic web services. In particular, the OWL-S proposal [40, 39] 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. The Guard-Stage-Milestone (GSM) approach [25, 17, 29] to artifact lifecycles permits services with pre- and post-conditions, parallelism, and hierarchy. Results in [17] show that the verification results described in the current paper can be generalized to apply to GSM. On a more practical level there is a close correspondence between business artifacts and cases as in Case Management systems [18, 52]; GSM captures most aspects of the conceptual models underlying those systems.

Static analysis of data-aware business processes. Work on formal analysis of artifact-based business processes in restricted contexts has been reported in [26, 27, 6]. Properties investigated include reachability [26, 27], general temporal constraints [27], and the existence of complete execution or dead end [6]. For the variants considered in each paper, verification is generally undecidable; decidability results were obtained only under rather severe restrictions, e.g., restricting all pre-conditions to be "true" [26], restricting to bounded domains [27, 6], or restricting the pre- and post-conditions to refer only to artifacts (and not their variable values) [27]. None of the above papers permit an underlying database, integrity constraints, or arithmetic.

[14] adopts an artifact model variation with arithmetic operations but no database. It proposes a criterion for comparing the expressiveness of specifications using the notion of *dominance*, based on the input/output pairs of business processes. Decidability is shown only by restricting runs to bounded length. [51] addresses the problem of the existence of a run that satisfies a temporal property, for a restricted case with no database, no arithmetic, and only propositional LTL properties. [3] considers another variety of the artifact model, with database but no arithmetic and no data dependencies, and limited modeling of the input from the environment. The work focuses on static verification of properties in a very powerful language (first order μ-calculus) which subsumes the temporal logic we consider (first order LTL), in particular allowing branching time. This expressivity comes at the cost of restricting verification decidability to the case when

the initial database contents are given. In contrast, we verify correctness properties for all possible initializations of the database.

Static analysis for semantic web services is considered in [44], but in a context restricted to finite domains.

More recently, [1] has studied automatic verification in the context of business processes based on Active XML documents. The works [22, 49, 2] are ancestors of [21] from the context of verification of electronic commerce applications. Their models could conceptually (if not naturally) be encoded as artifact systems, but they correspond only to particular cases of the model in [21]. Also, they limit artifact values to essentially come from the active domain of the database, thus ruling out external inputs, partially-specified services, and arithmetic.

Infinite-state systems. Artifact systems are a particular case of infinite-state systems. Research on automatic verification of infinite-state systems has recently focused on extending classical model checking techniques (e.g., see [13] 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 [9], rewriting systems with data [10, 8], Petri nets with data associated to tokens [36], automata and logics over infinite alphabets [12, 11, 45, 19, 32, 7, 8], and temporal logics manipulating data [19, 20]. However, the restricted use of data and the particular properties verified have limited applicability to the business artifacts setting.

2 Artifact Systems

We describe a minimalistic variant of the artifact systems model, adequate for illustrating our approach to verification. The presentation is informal, relying mainly on a running example (the formal development is provided in [21, 16]). The example, modeling an e-commerce process, features several characteristics that drive the motivation for our work.

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. 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 an attribute that uniquely identifies each tuple in a relation.

`PRODUCTS`(<u>`id`</u>, `price`, `availability`, `weight`),
`COUPONS`(<u>`code`</u>, `type`, `value`, `min_value`, `free_shiptype`),
`SHIPPING`(<u>`type`</u>, `cost`, `max_weight`),
`OFFERS`(<u>`prod_id`</u>, `discounted_price`, `active`).

The database also satisfies the following foreign keys:

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

Thus, the first dependency says that each `free_shiptype` value in the `COUPONS` relation is also a `type` value in the `SHIPPING` relation. The second inclusion 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` initialized to "edit_prod", and all other variables to "undefined". By convention, we model undefined variables using the reserved constant λ.

The services. Recall that artifacts evolve under the action of services. Each service is specified by a pre-condition π and a postcondition ψ, both 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, limited to linear inequalities over the rationals.

The following services 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.

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" $\wedge\ \texttt{prod\_id}' = \texttt{prod\_id} \wedge \texttt{ship\_type}' = \texttt{ship\_type})$

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 $\texttt{coupon}'$ 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 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 record.

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 Temporal Properties of Artifact Systems

The properties we are interested in verifying are expressed in a first-order extension of linear temporal logic called LTL-FO. This is a powerful language, fit to capture a wide variety of business policies implmented by a business process. For instance, in our running example it allows us 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.

In order to specify 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 [47]). 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,

[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.

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 $\mathbf{G}(p \rightarrow \mathbf{F}q)$, 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 $\texttt{status} =$ "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 end.

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.

Note that this property holds only due to the integrity constraints on the schema. Indeed, observe that (i) is guaranteed by the post-condition of service **choose_product**, (ii) by **choose_shiptype**, and (iii) by **apply_coupon**. In the post-conditions, the checks are perfomed by looking up in the database the weight/price/cost/limit attributes associated to the customer's selection of product id and shipment type (stored in artifact variables). The property performs the same lookup in the database, and it is guaranteed to retrieve the same tuples only because product id and shipment type are keys for PRODUCTS, respectively SHIPPING. The verifier must take these key declarations into account, to avoid generating a spurious counter-example in which the tuples retrieved by the service post-conditions are distinct from those retrieved by the property, despite agreeing on product id and shipment type.

Other applications of verification. As discussed in [21], 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 [6] 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, which can also be reduced to a verification problem.

Redundant attributes. Another design simplification consists of redundant attribute removal, a problem also raised in [6]. This is formulated as follows. We would like to test whether there is a way to satisfy a property φ of runs without using one of the attributes. This easily reduces to a verification problem as well.

Verifying termination properties. In some applications, one would like to verify properties relating to termination, e.g. reachability of configurations in which no service can be applied. Note that one can state, within an LTL-FO property, that a configuration of an artifact system is blocking. This allows reducing termination questions to verification.

4 Automatic Verification of Artifact Systems

Verification without constraints or dependencies We consider first artifact systems and properties without arithmetic constraints or data dependencies. This case was studied in [21], 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 1. *It is decidable, given an artifact system $\mathcal{A}$ 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 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.

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 a PSPACE verification algorithm.

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 [21] 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 [48] using automata-theoretic techniques. In both cases, the worst-case complexity remains PSPACE.

Verification with arithmetic constraints and data dependencies. Unfortunately, Theorem 1 fails even in the presence of simple data dependencies or arithmetic. Specifically, as shown in [21, 16], 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. Therefore, 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 [41]. 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 un unbounded number of times. To prevent such computations, the restriction imposed in [16] 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 [16] 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 [16] 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.

There is evidence that the feedback freedom condition is permissive enough to capture a wide class of applications of practical interest. Indeed, this is confirmed by numerous examples of practical business processes modeled as artifact systems, encountered in our collaboration with IBM. Many of these, including typical e-commerce applications, satisfy the feedback freedom condition. The underlying reason seems to be that business processes are usually not meant to "chain" an unbounded number of tasks together, with the output of each task being input to the next. Instead, the unboundedness is usually confined to two forms, both consistent with feeback-freedom:

1. Allowing a certain task to undo and retry an unbounded number of times, with each retrial independent of previous ones, and depending only on a context that remains unchanged throughout the retrial phase. A typical example is repeatedly providing credit card information until the payment goes through, while the order details remain unchanged. Another is the situation in which an order is filled according to sequentially ordered phases, where the customer can repeatedly change her mind within each phase while the input provided in the previous phases remains unchanged (e.g. changing her mind about the shipment type for the same product, the rental car reservation for the same flight, etc.)
2. Allowing a task to batch-process an unbounded collection of inputs, each processed independently, within an otherwise unchanged context (e.g. sending invitations to an event to all attendants on the list, for the same event details).

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. *[16] 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 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 1. Because of the data dependencies and arithmetic constraints, the isomorphism types of symbolic runs are no longer sufficient, 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[2] bounded by k^2, where k is the number of attributes of the artifact. 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.

Complexity and implementation. Unfortunately, there is a price to pay for the extension to data dependencies and arithmetic in terms of complexity. Recall that verification without arithmetic constraints or data dependencies can be done in PSPACE. In

[2] 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 [37].

contrast, the worst-case complexity of the extended verification algorithm is very high – hyperexponential in k^2 (i.e. a tower of exponentials of height k^2). The complexity is more palatable when the clusters of artifact attributes that are mutually dependent are bounded by a small w. In this case, the complexity is hyperexponential in w^2. If the more stringent acyclicity restriction holds, the complexity drops to 2-EXPTIME in k.

What does the high complexity say in terms of potential for implementation? As has often been noted, many static analysis tasks that have very high worst-case complexity turn out to be amenable to implementation that is reasonably efficient in most practical cases. Whether this holds for verification of business artifacts can only be settled by means of an implementation, which is currently in progress. Previous experience provides some ground for optimism. Indeed, a successful implementation of a verifier for Web site specifications that are technically similar to business artifacts without data dependencies or arithmetic is described in [23]. Despite the theoretical PSPACE worst-case complexity, the verifier checks properties of most specifications extremely efficiently, in a matter of seconds. This relies on a mix of database optimization and model checking techniques. We plan to build upon this approach in our implementation of a verifier for business artifacts.

5 Conclusions

We presented recent work on automatic verification of data-centric business processes, focusing on a simple abstraction of IBM's business artifacts. Rather than using general-purpose software verification tools, we focused on identifying resticted but practically relevant classes of business processes and properties for which fully automatic verification is possible. The results suggest that significant classes of data-centric business processes may be amenable to automatic verification. While the worst-case complexity of the verification algorithms we considered ranges from PSPACE to non-elementary, previous experience suggests that the behavior may be much better for realistic specifications and properties. We plan to further explore the practical potential of our approach using real-world business artifact specifications made available through our collaboration with IBM. More broadly, it is likely that the techniques presented here may be fruitfully used with other data-aware business process models, beyond artifact systems.

References

1. Abiteboul, S., Segoufin, L., Vianu, V.: Static analysis of active XML systems. ACM Trans. Database Syst. 34(4) (2009)
2. Abiteboul, S., Vianu, V., Fordham, B.S., Yesha, Y.: Relational transducers for electronic commerce. JCSS 61(2), 236–269 (2000), Extended abstract in PODS 1998
3. Hariri, B.B., Calvanese, D., De Giacomo, G., De Masellis, R., Felli, P.: Foundations of relational artifacts verification. In: Rinderle, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896. Springer, Heidelberg (2011)
4. Bhattacharya, K., Caswell, N.S., Kumaran, S., Nigam, A., Wu, F.Y.: Artifact-centered operational modeling: Lessons from customer engagements. IBM Systems Journal 46(4), 703–721 (2007)

5. Bhattacharya, K., et al.: A model-driven approach to industrializing discovery processes in pharmaceutical research. IBM Systems Journal 44(1), 145–162 (2005)
6. Bhattacharya, K., Gerede, C.E., Hull, R., Liu, R., Su, J.: Towards formal analysis of artifact-centric business process models. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 288–304. Springer, Heidelberg (2007)
7. Bojanczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., David, C.: Two-variable logic on words with data. In: LICS, pp. 7–16 (2006)
8. Bouajjani, A., Habermehl, P., Jurski, Y., Sighireanu, M.: Rewriting systems with data. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) FCT 2007. LNCS, vol. 4639, pp. 1–22. Springer, Heidelberg (2007)
9. Bouajjani, A., Habermehl, P., Mayr, R.: Automatic verification of recursive procedures with one integer parameter. Theoretical Computer Science 295, 85–106 (2003)
10. Bouajjani, A., Jurski, Y., Sighireanu, M.: A generic framework for reasoning about dynamic networks of infinite-state processes. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 690–705. Springer, Heidelberg (2007)
11. Bouyer, P.: A logical characterization of data languages. Information Processing Letters 84(2), 75–85 (2002)
12. Bouyer, P., Petit, A., Thérien, D.: An algebraic approach to data languages and timed languages. Information and Computation 182(2), 137–162 (2003)
13. Burkart, O., Caucal, D., Moller, F., Steffen, B.: Verification of infinite structures. In: Handbook of Process Algebra, pp. 545–623. Elsevier Science, Amsterdam (2001)
14. Calvanese, D., De Giacomo, G., Hull, R., Su, J.: Artifact-centric workflow dominance. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 130–143. Springer, Heidelberg (2009)
15. Chao, T., et al.: Artifact-based transformation of IBM Global Financing: A case study. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 261–277. Springer, Heidelberg (2009)
16. Damaggio, E., Deutsch, A., Vianu, V.: Artifact systems with data dependencies and arithmetic. In: Int'l. Conf. on Database Theory, ICDT (2011)
17. Damaggio, E., Hull, R., Vaculin, R.: On the equivalence of incremental and fixpoint semantics for business artifacts with guard-stage-milestone lifecycles. In: Rinderle, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896. Springer, Heidelberg (2011)
18. de Man, H.: Case management: Cordys approach. BP Trends (February 2009), `http://www.bptrends.com`
19. Demri, S., Lazić, R.: LTL with the Freeze Quantifier and Register Automata. In: LICS, pp. 17–26 (2006)
20. Demri, S., Lazić, R., Sangnier, A.: Model checking freeze LTL over one-counter automata. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 490–504. Springer, Heidelberg (2008)
21. Deutsch, A., Hull, R., Patrizi, F., Vianu, V.: Automatic verification of data-centric business processes. In: ICDT, pp. 252–267 (2009)
22. Deutsch, A., Sui, L., Vianu, V.: Specification and verification of data-driven web applications. JCSS 73(3), 442–474 (2007)
23. Deutsch, A., Sui, L., Vianu, V., Zhou, D.: A system for specification and verification of interactive, data-driven web applications. In: SIGMOD Conference (2006)
24. Dong, G., Hull, R., Kumar, B., Su, J., Zhou, G.: A framework for optimizing distributed workflow executions. In: Connor, R.C.H., Mendelzon, A.O. (eds.) DBPL 1999. LNCS, vol. 1949, pp. 152–167. Springer, Heidelberg (2000)
25. Hull, R., et al.: Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In: Proc. of 7th Intl. Workshop on Web Services and Formal Methods, WS-FM, pp. 1–24 (2010)

26. Gerede, C.E., Bhattacharya, K., Su, J.: Static analysis of business artifact-centric operational models. In: IEEE International Conference on Service-Oriented Computing and Applications (2007)
27. Gerede, C.E., Su, J.: Specification and verification of artifact behaviors in business process models. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 181–192. Springer, Heidelberg (2007), http://www.springerlink.com/content/c371144007878627
28. Glushko, R.J., McGrath, T.: Document Engineering: Analyzing and Designing Documents for Business Informatics and Web Services. MIT Press, Cmabridge (2005)
29. Hull, R., Damaggio, E., De Masellis, R., Fournier, F., Gupta, M., Heath III, F., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P., Vaculín, R.: Business artifacts with guard-stage-milestone lifecycles: Managing artifact interactions with conditions and events. In: ACM Intl. Conf. on Distributed Event-based Systems, DEBS (2011)
30. Hull, R., Llirbat, F., Kumar, B., Zhou, G., Dong, G., Su, J.: Optimization techniques for data-intensive decision flows. In: Proc. IEEE Intl. Conf. on Data Engineering (ICDE), pp. 281–292 (2000)
31. Hull, R., Llirbat, F., Simon, E., Su, J., Dong, G., Kumar, B., Zhou, G.: Declarative workflows that support easy modification and dynamic browsing. In: Proc. Int. Joint Conf. on Work Activities Coordination and Collaboration (1999)
32. Jurdzinski, M., Lazić, R.: Alternation-free modal mu-calculus for data trees. In: LICS, pp. 131–140 (2007)
33. Kumaran, S., Liu, R., Wu, F.Y.: On the duality of information-centric and activity-centric models of business processes. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 32–47. Springer, Heidelberg (2008)
34. Kumaran, S., Nandi, P., Heath, T., Bhaskaran, K., Das, R.: ADoc-oriented programming. In: Symp. on Applications and the Internet (SAINT), pp. 334–343 (2003)
35. Küster, J., Ryndina, K., Gall, H.: Generation of BPM for object life cycle compliance. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 165–181. Springer, Heidelberg (2007)
36. Lazić, R., Newcomb, T., Ouaknine, J., Roscoe, A., Worrell, J.: Nets with tokens which carry data. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 301–320. Springer, Heidelberg (2007)
37. Libkin, L.: Elements of Finite Model Theory. Springer, Heidelberg (2004)
38. Liu, R., Bhattacharya, K., Wu, F.Y.: Modeling business contexture and behavior using business artifacts. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 324–339. Springer, Heidelberg (2007)
39. Martin, D., et al.: OWL-S: Semantic markup for web services, W3C Member Submission (November 2003)
40. McIlraith, S.A., Son, T.C., Zeng, H.: Semantic web services. IEEE Intelligent Systems 16(2), 46–53 (2001)
41. Minsky, M.L.: Computation: finite and infinite machines. Prentice-Hall, Inc., Upper Saddle River (1967)
42. Nandi, P., et al.: Data4BPM, Part 1: Introducing Business Entities and the Business Entity Definition Language (BEDL) (April 2010), http://www.ibm.com/developerworks/websphere/library/techarticles/1004_nandi/1004_nandi.html
43. Nandi, P., Kumaran, S.: Adaptive business objects – a new component model for business integration. In: Proc. Intl. Conf. on Enterprise Information Systems, pp. 179–188 (2005)
44. Narayanan, S., McIlraith, S.: Simulation, verification and automated composition of web services. In: Intl. World Wide Web Conf, (WWW 2002) (2002)

45. Neven, F., Schwentick, T., Vianu, V.: Finite State Machines for Strings Over Infinite Alphabets. ACM Transactions on Computational Logic 5(3), 403–435 (2004)
46. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal 42(3), 428–445 (2003)
47. Pnueli, A.: The temporal logic of programs. In: FOCS, pp. 46–57 (1977)
48. Segoufin, L., Torunczyk, S.: Automata based verification over linearly ordered data domains. In: Int'l. Symp. on Theoretical Aspects of Computer Science, STACS (2011)
49. Spielmann, M.: Verification of relational transducers for electronic commerce. JCSS 66(1), 40–65 (2003), Extended abstract in PODS 2000
50. Wang, J., Kumar, A.: A framework for document-driven workflow systems. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 285–301. Springer, Heidelberg (2005)
51. Zhao, X., Su, J., Yang, H., Qiu, Z.: Enforcing constraints on life cycles of business artifacts. In: TASE, pp. 111–118 (2009)
52. Zhu, W.-D., et al.: Advanced Case Management with IBM Case Manager. Published by IBM,
`http://www.redbooks.ibm.com/redpieces/abstracts/sg247929.html?Open`