A Quest for Beauty and Wealth (or, Business Processes for Database Researchers) *

Daniel Deutch
Ben Gurion University
deutchd@cs.bgu.ac.il

Tova Milo
Tel Aviv University
milo@cs.tau.ac.il

ABSTRACT

While classic data management focuses on the data itself, research on Business Processes considers also the *context* in which this data is generated and manipulated, namely the processes, the users, and the goals that this data serves. This allows the analysts a better perspective of the organizational needs centered around the data. As such, this research is of fundamental importance.

Much of the success of database systems in the last decade is due to the beauty and elegance of the relational model and its declarative query languages, combined with a rich spectrum of underlying evaluation and optimization techniques, and efficient implementations. This, in turn, has lead to an economic wealth for both the users and vendors of database systems. Similar beauty and wealth are sought for in the context of Business Processes. Much like the case for traditional database research, elegant modeling and rich underlying technology are likely to bring economic wealth for the Business Process owners and their users; both can benefit from easy formulation and analysis of the processes. While there have been many important advances in this research in recent years, there is still much to be desired: specifically, there have been many works that focus on the processes behavior (flow), and many that focus on its data , but only very few works have dealt with both. We will discuss here the important advantages of a holistic flow-and-data framework for Business Processes, the progress towards such a framework, and highlight the current gaps and research directions.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: [Languages]; F.4.3 [**Theory of Computation**]: [Formal Languages]

*This work has been partially funded by the Israel Science Foundation, by the US-Israel Binational Science Foundation, by the EU grant MANCOOSI and by the ERC grant Webdam, agreement 226513.

PODS'11, June 13–15, 2011, Athens, Greece.

General Terms

Algorithms, Languages, Theory

1. INTRODUCTION

Recent years have seen s shift in data management research. While research on stand-alone database management continues to be foundational, increasing attention is directed towards the *context* in which this data is generated and manipulated, namely *the processes, the users, and the goals* that this data serves. In a nutshell, this is what research on Business Processes is all about.

A Business Process (BP for short) consists of a group of business activities undertaken by one or more organizations in pursuit of some particular goal. Such Business Processes describe the operational logic of applications, including the possible *application flow* and the *data* that is manipulated by it. Research in this direction involves marrying ideas and concepts from database management, with ideas from workflow and process flow management, that previously have been mostly studied separately. This combined flow-and-data management in Business Processes is the focus of this paper. We note that we use here the term "Business Processes" in a very broad sense and the problems and solutions that we will consider are applicable to many other contexts that involve data and flow, such as scientific workflows, Web applications and services, e-government and electronic patient records.

Consider a classic example of an inventory management of a company, say one that sells computer parts. Research that focuses only on the inventory data, its storage and manipulation (perhaps in a distributed setting, or even in the cloud) can tell us how to design the database, and how to query it in an optimal way. However, in the broader perspective of the company, the inventory database is only a tool that is used in the company *Business Process*, which, to the company, is itself of no less interest than the data. For instance, the company may provide an online interface for ordering computer parts. In this case, the interaction with the database may be initiated by an order made by an online user; the process in charge of processing of this order will query the database to find the required product, and if absent will issue an order from the mainline factory. If the product does not exist, the process may issue other queries on the products database, possibly to recommend the user other alternative products. The user may then choose one of these options or ask for more recommendations, which will be computed using a refined query, and so on.

Knowledge about the operation of such a Business Pro-

cess is thus of tremendous importance to both the company (or application) owners and their users. First, the process specification, and its possible executions flows, are themselves a valuable data; their analysis can be used by the application owners for detecting bugs or optimizing the process. For instance, in our computers company example, the company owners may wish to verify that whenever a user asks for a product that is not available, the appropriate actions are taken. Similarly, the application users may wish to infer the optimal ways of using the process, e.g. shortest (click-wise) navigation in the web-site to buy a computer with some specified features, or one that leads to minimal cost. Second, execution traces (logs) of Business Processes, detailing (parts of) the course of past execution, are another extremely valuable source of data. For example, logs analysis can reveal that customers are unsatisfied with the alternative products that are recommended to them, (as they typically navigate away from the store interface when presented with the suggestions), thereby helping to improve the application design. A combined analysis that uses both the process specification and its execution traces may also be extremely valuable. Tt may allow, for instance, to obtain answers to questions such as "what other alternative actions could users whose request failed take, but didn't". This may be used to improve the UI and make these options more apparent.

We observe that in the Business Process settings there is an inherent coupling of application *data* and *logic* (also referred to as *flow*): the flow affects the data and vice versa. This attracted the attention of database researchers (e.g. [10, 61, 46, 43, 32, 42]), that attempt to extend the good principles of data management (declarative specifications, optimization, etc.), to the broader perspective that includes not only the database itself but also the process surrounding it. Specifically, much of the success of database systems in the last decade is due to the beauty and elegance and of the relational model and its declarative query languages, combined with a rich spectrum of underlying evaluation and optimization techniques, and efficient implementations. This, in turn, has lead to an economic wealth for both the users and vendors of database systems. Similar beauty and wealth are sought for in the context of Business Processes. What is desired for Business Processes is an equivalently elegant formal model and query languages, that will allow easy formulation, readability and seamless optimizations, while at the same time will be expressively rich and comprehensive enough to capture intricate processes flow and their manipulation of data. Much like the case for database research, such beauty and richness of the model are also likely to result in economic wealth, for both the owners and the users of the processes.

While there have been many important advances in this research in recent years, there is still much to be desired. Specifically, there are several good models and techniques for capturing and analyzing the processes flow, and several good models and techniques for capturing and analyzing the data they manipulate. But the former often consider data only in a limited way while the latter typically capture the flow only in an implicit, data driven, manner. We are still missing a satisfactory, comprehensive solution for the *explicit* modeling and analyzing the processes flow *and* data, and their interactions.

We next briefly review the main tasks that are required towards achieving this goal, in the order considered in the rest of this paper. These tasks include *modeling* the processes, *generating* an instance of the chosen model, for a given Business Process, and *analyzing* (*querying*) process specifications and executions. The goal of the present paper is to highlight some of the main challenges in this area, review the state-of-the-art of research that tackles these challenges, and pinpoint several current research directions and challenges.

As always with such problems, the first challenge is *modeling*. One difficulty that rise when attempting an effective management of Business Processes is the typical complexity of their representations. Business processes usually operate in a cross-organizational, distributed environment and the software implementing them is fairly complex. For instance, our computer components company may interact in intricate ways with various suppliers of components, with the users that are placing orders, etc. But much like in the case of data management, effective Business Process management needs to rely on an abstract model for the Business Process. To answer this need, many different abstract representation models have been suggested (e.g. [5, 11, 12, 27, 10, 50, 33] and many others), that combine, to some extent, "classic" flow models such as state machines and context free systems with models that describe the underlying databases of the application and their interaction with flow. We review such models in Section 2.

Clearly, for Business Processes implemented in a language such as Java, their representation in such abstract models is not an easy task. Luckily, this gap is slowly being bridged by *declarative standards* that facilitate the design, deployment, and execution of BPs. In particular, the recent BPEL [14] standard (Business Process Execution Language), provides an XML-based language to describe the interface between the participants in a process, as well as the full operational logic of the process and its execution flow. The BPEL standard is relatively declarative, and it is much easier to abstract it further, to fit the model; there are also many graphical interfaces and editors that allow for a relatively simple, intuitive design of BPEL specifications.

Graphical interfaces are very useful in promoting a high-level (formal) specification of a Business Process. Still, in many cases, the specification of the Business Process is partially or completely unknown. In such cases, the abstract model needs to be *mined* [61, 66, 46] automatically, e.g. from a set of observed executions. In Section 3, we review both facets (graphical design interfaces and mining techniques) of obtaining instances of process specifications.

Once a process model is given, it may be used for *process querying and analysis*. Two kinds of analysis are of interest: the first is analysis of the possible *future* executions of a given process specification and the second is the analysis of *past* execution, stored in *execution logs*.

First, consider analysis of possible future executions (also referred to as *static analysis*). The idea is that we are given a specification of the Business Process (that is considered as our input data), and we wish to verify that some property (query) holds for *all* possible executions of the process. For instance, for our computer components company example, we may be interested in verifying that no customer can place an order without paying; or that whenever an order is processed, the system checks for the existence of the product in the database before asking an external vendor. Alternatively, we also be interested in computing the probability

that some property holds in a random execution, e.g. what is the probability for a user to ask for a component that is not in stock. The results of such analysis are of interest for both the process owners, as well as their users: the former may optimize their business processes, reduce operational costs, and ultimately increase competitiveness. The latter may allow users to make an optimal use of it (e.g. find out what is the most popular way of buying a computer?). Importantly, the results of such an analysis, that is aware not only of the company database, but also to the applications that manipulate it, can be very useful in improving the Database design, and in optimizing the data accesses (queries) made in the course of the Business Processes.

Next, let us consider the analysis of past executions. Execution logs detailing such past executions are of tremendous importance for companies, since they may reveal *patterns* in the behavior of the users (e.g. "users that buy a specific brand of RAM also tend to buy a specific brand of Motherboard"), may allow to identify run-time bugs that occurred, or a breach of the company policy etc. But the challenges are that typical repositories of such execution logs are of very large size, and that the patterns that are of interest are those that occur frequently but are not always known in advance. Consequently, various works considered data mining and OLAP (On-line Analytical Process) techniques for querying logs repositories; but there are also some works that introduce query languages for execution traces. Continuing the analogy with Databases, what is required for specifying the analysis tasks is a *declarative query language*, supported by efficient *query evaluation mechanisms*.

One particular challenge in the context of Business Processes and workflows is that of *security / privacy* [27, 59]. While the application owner may wish to disclose some of the process specification / logs (or parts of them) of the Business Processes, some (parts of) logs must be kept confidential, such as the credit card details of users performing a computer components purchase, or part of her interaction with the company interface, etc. Privacy/security have been studies (separately) for workflows and Databases. But what is desired is *combined* flow-and-data privacy/security mechanisms. In Section 4 we consider the analysis and querying of Business Processes, including all of the above aspects.

Personal Perspective. In a series of works (see e.g. [10, 28, 29, 30, 31]), the authors of this paper and collaborators have studied a particular model for Business Processes, its properties and the means for querying and analyzing it. Wherever in this paper we use the term "Our model", we refer to this model, initially defined in [10] and extended in further works.

2. MODELING

There are various models for Business Processes that may be found in the literature. Similarly to process models in verification and model checking (see e.g. [37, 20, 55, 53, 21, 58, 15] and many others), these models differ in their expressive power, i.e. in which sets of possible executions they can represent. But unlike works on verification, research on Business Processes further incorporates an explicit modeling of the *data* that underly the process. However, as we show below, in the vast majority of the models that consider data, the flow description is either implicit, or lacks expressive power. *In our opinion, what is critically missing is an integrated, expressive and intuitive (explicit) model for both the flow and the underlying data.* Such a model, accompanied with appropriate analysis tools / query languages, can allow process analysts to deduce important knowledge on the process, and ultimately improve its performance.

We start our review by considering some common flow models that are considered in verification; we then show how research on Business Processes extends these models to account for underlying data, of two kinds: relational and semi-structured. Finally, we will highlight the main challenges still remaining in modeling Business Processes. We will demonstrate the models using our running example of a company that sells computer components.

2.1 Control Flow Models

In the context of verification, the focus is typically on modeling (and analyzing; see section 4) the *flow* of applications. The simplest model that is studied is that of a *Finite State Machine* (FSM) [48]; the FSM states model the logical states of the application, and transitions are dictated by (typically external) input. For instance, the logical flow of our computer components company may be modeled by a Finite State Machine where some of the states may e.g. be "wait for user order", "processing user order", "search database for product", etc. But finite state models have a quite limited expressive power from a theoretical perspective, and furthermore modeling real-life systems with them may be cumbersome, and require a large number of states. The latter problem is somewhat alleviated by using the model of Hierarchial State Machine (HSM) [11]. HSMs allow to define hierarchy of Finite State Machines, in a way resembling "function calls". That is, an HSM consists of a set of state machines, and one state machine may "invoke" another, meaning that the control flow moves to the invoked machine; while the execution in this machine reaches an accepting state, the control returns to the invoking location [1]. For our example, the processing of orders may be modeled as a "function", itself represented by a Finite State Machine, encapsulating all states in the sub-process of order management.

The hierarchy of state machines gives conciseness in representation, but not a stronger expressive power. In real-life processes, there is often the need to represent (context-free) *recursion*. For instance, our specification should enable recursive invocations of the ordering sub-process, to account for more and more orders. A more expressive model that accommodates recursion is that of *Recursive State Machine* (RSM) [8]. Like for HSMs, in RSMs state machines may invoke each other, but they can furthermore do so in a *recursive* manner. RSMs have strictly more expressive power than HSMs, and in particular can capture context free languages [8]. There are multiple variants of RSMs, differing in the number of *entries* (initial states) and *exits* (final states) of the state machines. The class of RSMs where the state machines have multiple exits is strictly more expressive than the class of single exit recursive state machine [8].

Most of the process models, while modeling parallelism at the specification level, consider every single execution as a sequence (path) of events. There are however some explicit formal models for parallel executions, such as Petri Nets

[1]We mention in this context that the notion of hierarchy, along with additional constructs, lies at the core of the StateCharts [47].

[58]. In the context of Business Processes, our model [10] captures parallelism in the executions, as follows. The specification of a process in this model is a set of DAGs, connected through (possibly recursive) implementation relation, which is a notion similar to that of invocation in Recursive State Machines. The implementation relation is one-to-many, i.e. for every activity there are many possible implementations, and one is chosen at run-time, dictating the course of execution. The possible choices of implementation are annotated by *guarding formulas*. These are logical formulas on external effects such as user choices, network state, server response time etc., that may dictate the course of execution. For instance, a guarding formula may have the form $UserChoice =$ "$ShowRAMs$", and this formula will hold if the user chose to view the brand of RAMs suggested by the computer components store. At run-time the truth value of these formulas will determine the chosen implementation for every activity. For instance, depending on the user choice (and thus the satisfaction of the above formula), the user will be presented with the available RAM brands. Nodes in these DAGs correspond to business activities, while the edges reflect their ordering. Unlike RSMs, executions are defined as the entire graph (in this case DAG) that is obtained. This allows to capture *parallelism* in the execution, reflected by its DAG structure: activities that have no directed path between them are assumed to occur in parallel. For instance, the business logic underlying the web-based interface of our company may dictate that in parallel to having the system wait for users orders, *advertisements* are injected to the screen. Using a DAG structure for the execution it is much easier to capture such parallelism.

We also mention that our model in [10] for BPs has strong connections with that of Context Free Graph Grammars (CFGGs) [60, 52, 26]. CFGGs, similarly to "classic" context free (string) grammars (CFGs), CFGGs are composed of non-terminals, terminals, and derivation rules. But unlike string CFGs, with CFGGs the righthand side of the derivation rules contain *graphs*, and consequently a grammar define a family of graphs, that are derived by the grammar. There are many variants of this model, different in the definition of whether the non-terminals correspond to nodes or (hyper)edges of the graph, in how the derived sub-graph is connected (referred to as the "connection relation" [52]), etc. Our model may be considered as specific, restricted case of CFGGs, where all graphs are DAGs; the counterpart of derivation rules is the possibly recursive implementation relation described above.

Probabilistic Variants. In the context of Business Processes, it is important to model the uncertainty that is inherent in the external effects that govern the executions. For example, the course of execution of our computers store Business Process depends on the choices of users, submitted through its web interface. The choices that users will make are unknown at the time of analyzing the process, but we may have some probabilistic knowledge on the behavior of users. The above mentioned models all have *probabilistic* counterparts, where the possible executions (derivations) are associated with probabilities of occurring in practice. The probabilistic counterpart of Finite State Machines is Markov Chains [57], where every transition is associated with a probability of occurring in practice. These probabilities are assumed to be independent (the Markovian assumption is that the probability of taking a transition in a given state is independent on the previously made transitions), thus the probability of an execution is simply the multiplication of the probabilities along its transitions. Similar extensions were studied for probabilistic context free grammars (see e.g. [54]), and Recursive Markov Chains (RMCs) [39] that are the counterpart of Recursive State Machines; in these extensions the Markovian property is also assumed.

While the independence assumption simplifies the models and allow for a more efficient analysis (See section 4), it is in some cases unrealistic when modeling Business Processes. In particular, when the transitions are governed by external effects such as user choices, the state of the network, server response time etc., it is unreasonable to assume their independence: for instance, users that choose a particular computer component is more likely to later choose compatible components. Consequently, our model for *Probabilistic Business Processes* [30] (which extends the non-probabilistic variant) allows to associate the guarding formulas dictating the execution course with probabilistic events, and these events may be inter-dependent to some bounded extent. More specifically, a choice may depend on the n previous choices for some constant n. The value of n is typically small in real-life Business Processes (e.g. the choices of users depend on their previous choices, but only on a small number of such choices).

A Little Bit of Data. Before considering full fledged modeling of data in Business Processes, we consider flow models that capture data but only to a small extent. One such example is in our own Business Process model where the *guarding formulas* that "guard" the possible implementations use data variable. The value of these variables determines the formulas satisfaction and thus the choice of implementation for each activity. This, in turn, dictates the course of execution. However the use of data here is limited: guarding formulas may ask for the value of a specific data item, but cannot e.g. issue full-fledged SQL queries on the underlying Database to decide on the choice of implementation.

Another example is *scientific workflows* where the interaction between flow and data is given an interesting twist. Scientific workflows are processes that are executed by scientists and composed of modules. Each module performs some scientific task and the modules interact by passing data between them. The workflow specification may be hierarchical, in the sense that a module may be composite and itself implemented as a workflow. The data that is modeled here includes the input and output of every module, but not the internal database or the way it is manipulated / queried by the modules.

2.2 Underlying Relational Data

The models that we have mentioned so far mainly focused on the *flow*, i.e. they allow to capture the control flow of a given process, but the data that is manipulated by the process was only modeled to a limited extent. A complementary line of research, in the databases area, resulted in the development of a variety of process models that are centered around data. Such models have appeared even before the term Business Processes emerged in research, such as the model of Relational Transducers [5] (and the follow-up work on ASM transducers [64]), that was shown to be very effective in capturing e-commerce applications. In relational

$$
\begin{array}{rcl}
Ord(user, prod) & +:- & InOrd(user, prod) \\
PayedOrd(user, prod) & +:- & InPayment(user, prod) \\
UnpaidOrd(user, prod) & +:- & Ord(user, prod) \texttt{ AND} \\
 & & \texttt{NOT } PayedOrd(user, prod) \\
OutReceipt(user, prod) & +:- & PayedOrd(user, prod) \texttt{ AND} \\
 & & InStock(prod)
\end{array}
$$

Figure 1: Relational Transducer Example

$$
\begin{array}{rcl}
ArtifactClass & : & ComputerOrder \\
componentName & : & string \\
componentType & : & "RAM" \\
processed & : & bool \\
PaidFor & : & bool
\end{array}
$$

Figure 2: Artifact Example

transducers, the state of the application is modeled as a relational database and the database state is modified using the repeated activation of queries (in the spirit of active databases [3]). The database is used to model not only the internal database state, but also the interaction of the transducer with the external environment: these are modeled via a set of input and output relations. There is thus no clear distinction between the *flow* and the *underlying data* - the database stores it all. A Datalog-like program is used to query and update the state, input and output relations. For instance, the rules in Figure 1 can be used to (partially) specify the business logic of our computers company. Relation names starting with "In" stand for input relations, those with "Out" for output relations, and the rest are state relations. Here the semantics of the rules is like in inflationary Datalog, that is, a satisfying assignment to the righthand side leads to an addition of a corresponding tuple for the relation whose name appears on the rule left hand side.

Note the process flow that is implicit in the above description: the input orders made by the user change the state of the orders, and user payments change the order status from unpaid to payed. *The difficulty is that the underlying flow is not easily observable from the process specification, especially in the common case when the number of state relations is large and inter-dependent.* As a consequence, the process designers and analysts may have difficulty grasping the causality and temporal relationship between actions/data, e.g. which part of the data is influencing other parts, which is updated before the other and how, etc. This is in contrast to case of the flow models mentioned above, where the execution flow is immediately observable from the specification.

A similar approach is taken in the works on Business Process artifacts [50, 42, 68, 2]. The model focuses on the data received, generated and manipulated by the process (e.g. purchase orders, sales invoices, etc.). The data structures encapsulating this information are referred to in this context as *artifacts*. An example for such an artifact, for a user order of a computer component is given in Figure 2.2. Artifact states are queried and modified by *services*, which are defined by declarative rules, accompanied by pre-and post conditions for their invocation. Again, there is an implicit modeling of flow, since the state of the order (artifact) reflects whether or not it was processed, and whether or not it was paid for, etc. and this information is updated as the flow evolves. But similarly to the case of relational transducers, the description of flow is somewhat implicit.

Last, we mention that the modeling of data-centered Web Applications have also received a significant attention in recent years. One notable model in this context is the one of data-driven, interactive Web Applications [32, 34] which are essentially implicitly specified *infinite-state machines* in which the state (database) is shared across various applications, each of which may read/write its contents. The interaction of web services, and specifically the data passed between them, were studied also in the context of [16].

2.3 Underlying Semi-structured Data

Before we conclude, we mention another class of works in this context that considers data which is not relational, but rather semi-structured, and specifically XML. Here again, there are works with an underlying flow model of a finite state machine or context free languages, as well as some probabilistic models. We next review them briefly. The Active XML model [1, 7] extends XML with Web Service calls, whose results are embedded back in the document, allowing to make additional calls etc. In a recent work [2] the authors suggest an artifacts model that combines a simple flow model of Finite State Machine, with the data model of Active XML. Also, recent works have studied models for probabilistic XML documents that is based on a generative process modeled by a Recursive State Machine [12]. In our opinion, these works are important steps towards rich models that combine flow and data; but more work is required to further find the correct balance between the models expressiveness, and the efficiency of query evaluation they allow (see Section 4).

2.4 Research Directions and Open Problems

We conclude this section by listing a few aspects that are absent in the current Business Processes and workflow models, yet we believe are important.

- Arguably the biggest challenge in Business Process modeling is the combination of rich flow model with rich model for underlying database manipulation. As we will see in the next section, high complexity or even undecidability of analysis is difficult to avoid whenever such rich models are considered. We note that the recent work of [2] has made a progress in this area by designing an artifact model for Active XML [7] with an underlying, explicitly modeled, Finite State Machines. But there is still a long way to go, extending the model expressivity within the boundaries of decidability.

- In [25] the authors state that no model is likely to be "the best" for all needs, and consequently that there is a need for the development of a theory (and practical implementations) of *views* on Business Processes and a practical mapping between them. We concur and believe that this is even more true for rich, combined flow and data model. Recently, there have been a few advancements in this respect. In [2] the authors sug-

gest the notion of a workflow view and use it as a way of comparing expressive power of workflow specification languages. We believe that there are important research challenges in the efficient computation and maintenance of such views.

3. GENERATION OF A MODEL INSTANCE

In the previous section we have considered the modeling of Business Processes, and discussed various possible choices for such models. Once such a model is chosen, we need to create an instance of this model for the given business process. Ideally, this would be done as part of the process specification, with the actual software that implements the process being automatically generated following the specification (and thus matching precisely the model). In this case, the challenge lies in suggesting effective specification tools, and in particular convenient visual interfaces for the designer. Alternatively, the process model can be defined manually, describing as close as possible the (already existing) Business Process, or be automatically mined from the available execution logs.

In recent years there has been significant effort in all these directions, in both industry and the academia. However, here too there is still much to be desired. We next review the state-of-the-art in these directions: we first consider manual model design, and then automatic mining tools.

3.1 (Manual) Instance Design

The increasing interest in high-level Business Process specifications has triggered research and development geared towards *User Interfaces* that will allow such process design. Many of these interfaces are based on process *visualization*. This has many advantages: it allows the designer to easily formulate and modify the process logic, allows users to easily understand the process logic and how to use it, and allows analyzers to easily pose analysis tasks. One standard class of such visual interfaces uses *UML (Unified Modeling Language) [13] state diagrams*. These are essentially StateCharts [47] with standardized notation. Such state diagrams allow to represent the process flow (using hierarchical states), along with the process interaction with its environment and other processes (i.e. events that affect the flow, messages that are emitted by the process during the flow, etc.) guarding the different transitions. While StateCharts (and consequently UML state diagrams) are highly successful as a user-friendly, graphical model for designers of process *flow*, the incorporation of *data* and its *interaction with flow* in this model may be quite intricate. In principal, this can come as part of the description of events actions that guard the transitions. But unlike the case for flow, the UML state diagrams standard does not dictate any syntax for the exact formulation of events and actions, and *the common practice is to use "structured English", or high-level programming languages such as C or Java (but not database query languages) for expressing the conditions and actions* [36].

Another standard for specifying Business Processes that has emerged in the last decade is the *BPEL (Business Process Execution language) standard*. This standard, developed jointly by BEA Systems, IBM, and Microsoft, combines and replaces IBMŠs WebServices Flow Language (WSFL) and MicrosoftŠs XLANG. It provides an XML-based language to describe, *in a declarative way*, the interface between the participants in a process, as well as the full operational logic of the process and its execution flow. Commercial vendors offer systems that allow to design BPEL specification via a visual interface, using a conceptual, intuitive view of the process as a graph; these designs are automatically converted to BPEL specifications, *and can further be automatically compiled into executable code that implements the described process*. Unlike StateCharts, BPEL specifications do allow some explicit representation of data. However this representation is mainly geared towards the modeling of the *interaction* of different processes (and more specifically web-services). It includes an XML schema for sent and received data (captured by variables), and an XPath interface for accessing these variables. But it has no explicit modeling of full-fledged database manipulation.

Figure 3 depicts an example BPEL specification, edited through a Graphical User Interface. The specification given here describes the operational logic of a fictive travel agency called "AlphaTours" (as may be observed from the property nodes on the upper righthand side of the figure). Observe that the *control flow* of the agency is captured via a DAG, appearing in the "behavior" tab; also for every activity, the *input data* type required by the activity and the *output data* type that it generates are depicted as nodes in the "Data" tab; these nodes are connected to the corresponding flow activities through unique "data edges". For instance, the input of the "searchTrip" activity is a trip request, and its output ("tripResults") is fed as input to the "reserveTrip" activity, etc.

Another line of tools that allows for declaratively specifying Business Processes / Web Applications flow and their underlying database include WebML and Web Ratio [22], WAVE [33], Hilda [70], Siena [24], and others. Here however the representation of flow is only implicit, and an intuitive flow visualization in the style described above is not available. *What is desired, and is the subject of an intriguing research is an interface for explicit visualizing of both the processes flow and manipulation of data*. Of course, there may be parts of the specification (both in terms of flow and data manipulation) which are complicated and difficult to visualize in a user-friendly form. However, it is still worthwhile to intuitively visualize those parts that are simpler, and perhaps suggest simplified "views" on the more complicated part.

3.2 Mining Model Instance

Even the best interface for process design requires a designer that will use it to specify the logic of the process. Works on *process mining* take an orthogonal approach. It tries to infer the process structure from a set of observed (perhaps partial) executions. Even if we consider only flow, this is clearly a challenge: consider a set of logs containing many different operations, occurring in different orders among different logs. Inferring a process model here includes deciding which of these operations are defined to occur in "parallel" (i.e. in an arbitrary order), and which of them are in order. But it may be the case that for two activities which may actually be parallel in the process specification, there exists some consistent ordering in the given set of logs - simply because the number of "sampled" (observed) logs is not big enough to exhibit all orderings. More generally, identifying the causality relationship between the invocation of activities is a major challenge in this context.

There are several approaches for process mining, attempt-

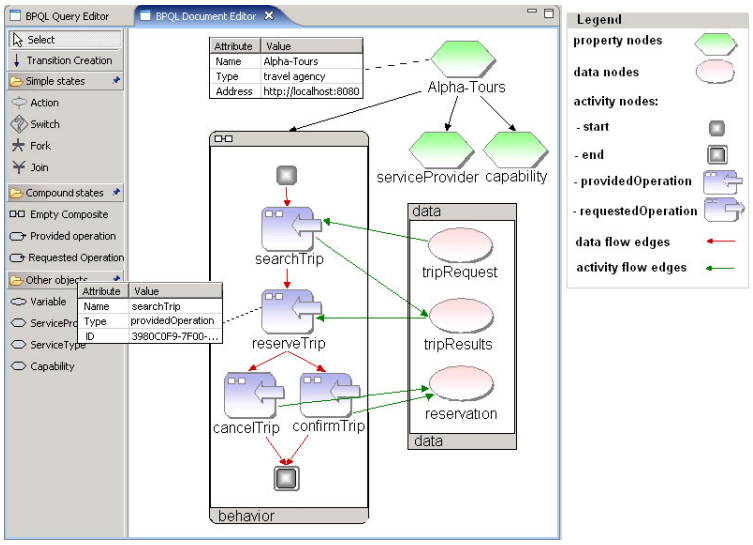

Figure 3: A process example modeled using BPEL Editor

ing to alleviate these difficulties. Some of these approaches use techniques such as Neural networks and Association rules mining for learning the process structure. This includes the Process mining tools in the Business (Process) Intelligence (BPI) suite [46]. While these approaches have some experimentally proven successful, it is difficult to provide any formal guarantees on their results. Other approaches [62, 54] are based on statistics, and try to maximize the likelihood of the observed sequence given the model. These approaches are rooted in works on formal models, such as those on inference of a best fit (i.e. maximizing the likelihood of observations, given the model) Markov Chain [62], or a Probabilistic Context Free Grammar [54] for a given set of observations. However these models typically do not model *parallelism*, which is inherent in process models. Moreover, to the best of our knowledge, these approaches all pertain to the application structure and flow, some of them in the broader context of the organization and the interaction of the process with it and with other processes, *but not the manipulated data*.

3.3 Research Directions and Open Problems

To conclude this section, we consider some open questions and promising research directions in this context of the works presented above.

- Similarly to the case with process modeling (and thus not surprisingly), visual interfaces for designing processes flow hardly provide tools for modeling the data that is manipulated. Of course, one can design externally a database schema and simply annotate the flow with SQL queries. But still, for instance, visualizing (even some part of) the effect of the data on the flow, and vice versa, is an intriguing and non-trivial task. What is required for that is an holistic model and visual interface for both flow and data. The lack of treatment of data is even more noticable in the works on process mining, as observed above; any advance towards this end would be very important.

- The declarative nature of some interfaces for process design, such as BPEL, opens the way for dedicated *optimization* techniques, improving the performance of the processes. There have been works on optimization of the execution of declaratively described processes, in various contexts (e.g. [69, 63]). However, there is currently no theory of declarative Business Process optimization, that is comparable to the comprehensive theory on database query optimizations.

- Due to the difficulty of process mining, one can consider an hybrid approach, where some parts of (or "clues about") the process specification are given, but other parts need to be inferred from the observed executions. This could model relatively well real-life scenarios.

4. QUERYING

As exemplified in the Introduction, Business Processes serve as an important mine of information for the processes owners as well as their users. There are (at least) two kinds of analysis that are of interest in this context: first, the analysis of possible (and, perhaps, probable) *future* executions of the process. This can be e.g. used to preempt possible bugs or breach of policies in future executions. Second, querying logs of *past* executions can also reveal important information, such as problems that occurred at runtime, trends in the process usage, etc. To support such an analysis, what is desired is a formal *query language*.

Desiderata. We next list some of the desired features of such a query language.

- It is desired that such query language will be, like the Business Process specification itself, *declarative*, *intuitive* and *graphical*, allowing the process designer to specify the analysis tasks side by side with the design of the process specification.

- The same query language will preferably allow uniformly for the analysis of future as well as past execution. Such uniformity will have several advantages: it will require analysts to only master a single language, it will allow to combine specification analysis with log querying, it may allow for uniform development of query optimization techniques, etc.
- We have stressed the importance of models that describe both the process flow and its underlying data. Correspondingly, the query language that is used to analyze the modeled Business Processes should allow to query *both the process flow and the data*, and the interactions between them.
- The query language should allow to pose queries at different levels of granularity, specified by the analysts. For instance, one may wish to ask coarse-grain queries that consider certain process components as black boxes and allow for high level abstraction, as well as fine-grained queries that "zoom-in" on all or some of the process components, querying there inner-workings.
- The query language should allow to specify "boolean" verification queries, pertaining to the existence of bugs, the enforcement of policies etc, as well as"selection" and "projection" queries that allow to retrieve some parts of the process or its executions, that are relevant for the analysis. For instance, we may not only be interested in the boolean existence of a bug, but if it exists we wish to know in which part of the process it occurred / may occur, under what circumstances, what is may affect, etc.
- Naturally, to make things practical, the query language should allow for efficient query evaluation algorithms and optimizations.

We next review the state-of-the-art in works on querying future and past executions of Business Processes, highlighting which of these desired properties were (partially) achieved, and which still remain as challenges.

4.1 Querying Future Executions

Given a Business Process specification, analysts are typically interested in testing/querying properties of its possible future executions. The property may be some constraint that is expected to hold in every execution, such as "a user can not place an order without giving her credit card details" or "a user must have a positive balance in her account before placing an order". It may also be the probability that some situation occurs in a random execution, such as "what is the probability that a user chooses to order a Motherboard given that she ordered a RAM?".

The analysis of possible future executions of a process, also referred to as *static analysis*, was extensively studied for formal flow models (Finite State Machines, Pushdown automata, Context Free Grammars, etc.). The dominant approach for such analysis is to use a *temporal logic* formalism [37] to query the possible executions of a given process. In temporal logic, formulas are constructed from predicates and temporal quantifiers such as "before" (B) ,"always" (A), "until" (U) etc. For instance, a query of the sort "a user must login *before* placing an order" can be expressed by a formula of the form $Login \ \ B \ \ Order$, where $Login$ and $Order$ are predicates that can be checked locally on a given flow state.

While temporal logic is very useful, it fails to satisfy many of the desiderata in the context of Business Processes. The main difficulties in using it for analysis of Business Processes executions are the following. First, similarly to the case of First Order Logic, the design of temporal logic formulas for complicated properties may be cumbersome, and the outcome may be difficult to read. Second, in temporal logic there is no explicit reference to underlying process data, and the properties it allows to capture pertain only to the process flow. Third, it allows only to verify boolean properties, and cannot express any counterpart of selection and projection database queries, and in particular does not allow to retrieve paths that are of interest.

The shortcoming of explicit constructs for referring to both the flow and data was alleviated by the work of [32], suggesting a query language called LTL-FO (for Linear Temporal Logic-First Order). The idea is that temporal logic is used for querying the execution flow, while First Order constructs may be used inside the predicates, for querying not only the execution flow state, but rather the *database* state at the current point of the execution. For instance, the following formula specifies that whenever a user orders any product, its balance must be positive (we assume that Order and Balance are database relations. A is the temporal operator "Always" and user, sum, product are used as bound variables in the formula).

- $\forall user, product. A(Order(user, product) \Rightarrow \exists sum > 0. balance(user, sum))$

LTL-FO is an important step in the direction of querying Business Processes. We also mention in this context the work on querying Active XML documents [4], introducing tree-LTL, where temporal operators are used over tree patterns. But these languages still lack some of the above desiderata: they do not allow to specify non-boolean queries, do not allow controlled levels of granularity, and they are hard to formulate for non (logic) experts.

Some of the desiderata that are not achieved by the different variants of temporal logic (with or without constructs for querying data), are satisfied by our query language BPQL (Business Process Querying Language) [10], allowing to pose queries on our Business Process model. BPQL is based on abstraction of the BPEL formalism, along with a graphical user interface that allows for simple formulation of queries on BPEL specifications. At the core of the BPQL language are Business Process *patterns* that allow users to describe the pattern of activities/data flow that are of interest. Business Process patterns are similar to the tree and graph-patterns offered by query languages for XML and graph-shaped data, but include two novel features designed to address the above desiderata. First, BPQL supports navigation along two axis: (1) the standard path-based axis, that allows to navigate through, and query, paths in process graphs, and (2) a novel zoom-in axis, that allows to navigate (transitively) inside process components and query them at any depth of nesting. Second, paths are considered first class objects in BPQL and can be retrieved, and represented compactly. An example for such an execution pattern, designed using the BPQL Graphical User Interface, is depicted in Figure 4.

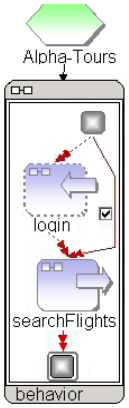

Figure 4: A query example modeled using BPQL Editor

This query looks for a login activity, followed, after some sequence (path) of choices, by a searchFlights activity.

Recall however that our Business Processes model does not allow explicit modeling of data, and correspondingly BPQL does not have explicit constructs for underlying data, but rather focuses on querying the execution flow of the process. More work is required in combining the desired features of BPQL, with the ability to formulate data-aware queries.

Complexity Results. We briefly review some main complexity results for the query formalisms described above. First, the complexity of evaluation of Linear Temporal Logic (LTL) queries (as well as branching-time logic CTL^* [37]) on Finite State Machines is known to be linear in the state machine size but exponential in the size of the formula. In [8] the authors show the same complexity results for context free processes, and equivalently for Single Exit RSMs [38]. Similarly, for BPQL, the complexity of query evaluation was shown to be polynomial in the process specification size, with the exponent depending on the query size. We have mentioned above the query language LTL-FO for querying both flow and data. In [35] the authors show that evaluating LTL-FO queries on their process model is *PSPACE-complete* under some restrictive assumptions on the input process, and is undecidable in general (although they describe in [32] a practical implementation based on strong heuristics). It is observable that the blowup in the complexity of query evaluation is difficult to avoid when querying flow and data together. Finding a framework that will satisfy all of the above listed desiderata, and will in particular allow for efficient query evaluation, is an important research challenge.

4.2 Querying Past Executions

We have discussed the analysis of future executions of a given process specification. Equally important is the analysis of executions that took place in the past, and are recorded in a repository. Execution logs are of tremendous importance for companies, since they may reveal *patterns* in the behavior of the users (e.g. "users that buy a specific brand of RAM also tend to buy a specific brand of Motherboard"), may allow to identify run-time errors that occurred, or a breach of the company policy. In the context of scientific workflows (see section 2), these execution logs are referred to as *provenance*. They represent instances of the scientific process that was used in practice, and they are in fact the main object that is analyzed in this context. This analysis can verify the correctness of experiments represented by the workflow, identify the different tools that were used, with which parameters, etc. Another application of log analysis is inference of probabilities, based on past observations, for the probabilistic process models discussed in section 2.

Contents of the Log. One basic question in this context is what is recorded in execution logs. One option is to record only the execution flow, i.e. the activities (functions, web-services, modules..) that were used at run-time, along with their order of occurrence. But in many cases what is also of interest is the record of *data* that was manipulated / transmitted throughout the execution, and the interaction of data with the execution flow. This entails "marrying" workflow provenance (that includes record of the activities or modules that occurred at run-time), with the notion of *data* provenance (e.g. [45, 40, 67, 17, 44, 41, 23, 19, 18, 71, 49]) - the management of fine-grained record on the course of databases queries evaluation.

There are two challenges in this context: first, keeping a complete record of all activities that occurred at run-time, and all data that was manipulated, may be infeasible in terms of the *required storage resources*. Second, while parts of the logs may be essential for analysis purposes, it is often the case that other parts of the logs should be kept confidential. How do we decide what to record?

There are various works that suggest partial solutions to (different aspects of) this challenge. In [28], we have suggested a selective tracing systems for Business Processes execution flows, that uses a *renaming function* for activities, to mask the real names of the recorded activities, and a *deletion set* for activities, allowing the deletion of all record of some activities from the execution logs. In the context of scientific workflows, there are recent works [6, 51] that suggest frameworks for keeping track of both the activated modules and their internal manipulation of data. But the provenance that is tracked there is not fine-grained "enough", in the sense that it does not keep track of the exact transformations performed on each data item (in contrast to the case of provenance management in "standard" databases, see e.g. [45]). In [27] the authors distinguish three types of workflow privacy: module privacy, data privacy, and provenance privacy. The first refers to hiding the functionality of one or more modules in a workflow, the second refers to hiding the internal data of a module, and the third refers to hiding the way a data item is manipulated and transmitted between the modules of the workflow. There are still many open challenges in workflow security, as described in [27].

What Formalisms to Use for Analysis. Ideally, the same query language used for querying future executions could be used for querying past executions, with the appropriate adaptation of the query evaluation algorithms. This may allow analysts to only master a single language, allow to combine specification analysis with log querying and allow a uniform development of query optimization techniques. Such uniformity may be challenging to achieve, since queries that can be evaluated on a given log (or a database of logs) may

become undecidable when evaluated statically over (potentially infinite number of) possible future executions. These challenges can possibly be addressed by a careful choice of the query formalism. For instance, we have shown in [28] that BPQL, originally designed for querying future executions, can be used also as query language for execution logs. LTL-FO, mentioned above in the context of querying future executions, can also be adapted for querying past executions. There are also other works (e.g. [56]) that show how to use temporal logic for analysis of execution logs. The similarities of solutions entail also similarities in the remaining challenges described above, and in particular the design of a user friendly (graphical) query language, that will allow to query both the flow and data of execution logs.

Challenges and (Partial) Solutions in Query Evaluation. One challenge in querying log repositories is that their typical size is very large (perhaps distributed among many locations, for complex distributed systems), as logs are collected over a long period. A first thing to observe is that process logs may be abstractly viewed as graphs that record the execution flow. Much research has been dedicated to query evaluation over graphs, developing various index structures and labeling schemes for the graph nodes to speed up query processing. There is naturally also large body of work on standard database query optimization. But, to our knowledge, the problem of efficient combined analysis of flow-and-data has not yet been addressed.

In the context of Business Processes, information about the process structure (derived from its specification) may be used to further improve performance. For example, in [28] we have presented a framework for type-based optimization of query evaluation on execution logs, based on the observation that the specification reveals useful information on the kinds of flow that can (and cannot) appear in the execution logs, and this information can be used to optimize query evaluation. Similarly, we have shown in a recent paper [9] that, using the process specification, one may derive compact labels for the graph nodes, thereby allowing for efficient processing of reachability queries. These works however focus on the flow and extending them to handle data aspects is intriguing.

An additional challenge is that in many cases what is sought for in these logs is vaguely defined as "interesting patterns", and is difficult to express formally. For these reasons, a dominating approaches is to use *data mining* and OLAP techniques for analyzing log repositories. Indeed, many works adapt "classic" data mining techniques such as clustering, associating rule mining and various sorts of statistical analysis [46]. One issue that appears uniquely in execution logs (in contrast to "general" data mining) is the *temporal* nature of events. To that end, there are dedicated data mining techniques such as *Sequential Patterns* and *Complex Event Processing* that account for the order in which events occur and find patterns in the sequence of events. But these works focus mainly on the execution flow, rather than on data. Here again what is missing is an effective processing accounting for both flow and data.

Run-time Monitoring. To conclude this section we briefly consider a third axis of analysis, namely that of *run-time monitoring* of the process execution. For instance, the owners of the computer store in our running example may be interested in being notified whenever a user places orders for five or more different products in, say, less than ten seconds (because this may indicate that a malicious automated process in fact placed these orders).

There have been several approaches to the runtime monitoring of Business Processes. One approach [46] is to use pre-defined templates (such as a process P was at a state S for more than X seconds), that can be instantiated (i.e. fed with concrete values) by analysts via a form. The instantiated tasks are internally implemented using e.g. SQL statements. But the templates are fixed and are intended to be designed only by the system administrator, and not by the analyst. This approach is effective in some cases, but fails to satisfy the desiderata we listed above. More in the spirit of our desiderata are works that lift the same formalisms used for static analysis of executions, and use them in the context of monitoring. Specifically, [56] showed how to use LTL for effective monitoring of distributed systems; this is an interesting direction but it suffers from the same problems discussed above regarding temporal logic. In [10] we show how to adapt BPQL for monitoring tasks, but again, the focus is on the process flow rather than its data. As before, we believe that what is missing is a comprehensive flow-and-data monitoring solutions, that will preferably be employed uniformly with a solution for the analysis of past and future executions.

4.3 Research Directions and Open Problems

To conclude, we list several research directions and open questions in the context of querying Business Processes.

- We have presented many different formalisms for querying processes, however most of them focus only on flow and not on data, and those that allow to query flow-and-data are computationally expensive in terms of worst case complexity. Further studying the tradeoff between expressivity and complexity of query evaluation in this context is an intriguing research direction.
- One specifically important benefit of declarative query languages is a seamless optimization. The development of dedicated optimization techniques for the analysis of past, present and future executions, perhaps based on the information given by the Business Process specification, is an important challenge.
- We have mentioned the work on provenance for scientific workflows and for database queries. As described above, maintaining fine-grained (i.e. flow and data) provenance for scientific (and other) workflows is useful, but may require a lot of storage space. Finding efficient ways of storing fine-grained provenance, and then utilizing it for analysis tasks, is the subject of an on-going research. We note in this context that economical storage of temporal databases has been extensively studied (see [65]), but the additional challenges here are due to the possibly intricate structure of the process *flow* that is absent from temporal databases, and entails intricate causality relations between process activities and data items.

5. CONCLUSION

We provide in this paper a database perspective to Business Processes. Research in this area considers the context

(processes) in which data is generated and manipulated, the users interacting with the data and the goals that the data serves. We have exemplified the importance of such a holistic approach to process and data management and its benefits. We have discussed the state-of-the-art research in this direction and highlighted the remaining gaps. We argue that similarly to the case of "conventional" database research, elegant modeling and rich underlying technology are likely to bring economic wealth for the Business Process owners and their users. Consequently, we believe that Business Processes constitute a very promising area for research and that Database researchers are best equipped to lead a breakthrough in this field.

6. REFERENCES

[1] Serge Abiteboul, Pierre Bourhis, Alban Galland, and Bogdan Marinoiu. The axml artifact model. In *TIME*, 2009.

[2] Serge Abiteboul, Pierre Bourhis, and Victor Vianu. Comparing workflow specification languages: a matter of views. In *ICDT*, 2011.

[3] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[4] Serge Abiteboul, Luc Segoufin, and Victor Vianu. Static analysis of active xml systems. *ACM Trans. Database Syst.*, 34(4), 2009.

[5] Serge Abiteboul, Victor Vianu, Brad Fordham, and Yelena Yesha. Relational transducers for electronic commerce. In *PODS*, 1998.

[6] Umut Acar, Peter Buneman, James Cheney, Jan Van Den Bussche, Natalia Kwasnikowska, and Stijn Vansummeren. A graph model of data and workflow provenance. In *TaPP*, 2010.

[7] Active XML. http://activexml.net/.

[8] R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. Reps, and M. Yannakakis. Analysis of recursive state machines. *ACM Trans. Program. Lang. Syst.*, 27(4), 2005.

[9] Zhuowei Bao, Susan B. Davidson, Sanjeev Khanna, and Sudeepa Roy. An optimal labeling scheme for workflow provenance using skeleton labels. In *SIGMOD*, 2010.

[10] C. Beeri, A. Eyal, T. Milo, and A. Pilberg. Monitoring business processes with queries. In *Proc. of VLDB*, 2007.

[11] M. Benedikt, P. Godefroid, and T. Reps. Model checking of unrestricted hierarchical state machines. In *ICALP*, 2001.

[12] Michael Benedikt, Evgeny Kharlamov, Dan Olteanu, and Pierre Senellart. Probabilistic XML via Markov chains. *PVLDB*, 3(1), September 2010.

[13] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language user guide*. Addison Wesley Longman Publishing Co., Inc., 1999.

[14] Business Process Execution Language for Web Services. http://www.ibm.com/developerworks/library/ws-bpel/.

[15] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30, 1983.

[16] T. Bultan, J. Su, and X. Fu. Analyzing conversations of web services. *IEEE Internet Computing*, 10(1), 2006.

[17] P. Buneman, J. Cheney, and S. Vansummeren. On the expressiveness of implicit provenance in query and update languages. *ACM Trans. Database Syst.*, 33(4), 2008.

[18] Peter Buneman, James Cheney, and Stijn Vansummeren. On the expressiveness of implicit provenance in query and update languages. In *ICDT*, 2007.

[19] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In *Proc. of ICDT*, 2001.

[20] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 1020 states and beyond. *Inf. Comput.*, 98(2), 1992.

[21] O. Burkart and B. Steffen. Model checking for context-free processes, 1992.

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

[23] James Cheney, Stephen Chong, Nate Foster, Margo I. Seltzer, and Stijn Vansummeren. Provenance: a future history. In *Proc. of OOPSLA*, 2009.

[24] David Cohn, Pankaj Dhoolia, Fenno Heath, Iii, Florian Pinel, and John Vergo. Siena: From powerpoint to web app in 5 minutes. In *ICSOC*, 2008.

[25] David Cohn and Richard Hull. Business artifacts: A data-centric approach to modeling business operations and processes. *IEEE Data Eng. Bull.*, 32(3):3–9, 2009.

[26] B. Courcelle. Recognizable sets of graphs, hypergraphs and relational structures : a survey. In *DLT '04*, volume 3340 of *LNCS*, 2004.

[27] Susan B. Davidson, Sanjeev Khanna, Sudeepa Roy, and Sarah Cohen Boulakia. Privacy issues in scientific workflow provenance. In *WANDS*, 2010.

[28] D. Deutch and T. Milo. Type inference and type checking for queries on execution traces. In *Proc. of VLDB*, 2008.

[29] D. Deutch and T. Milo. Evaluating top-k queries over business processes (short paper). In *Proc. of ICDE*, 2009.

[30] D. Deutch and T. Milo. Top-k projection queries for probabilistic business processes. In *Proc. of ICDT*, 2009.

[31] D. Deutch, T. Milo, N. Polyzotis, and T. Yam. Optimal top-k query evaluation for weighted business processes. In *Proc. of VLDB*, 2010.

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

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

[34] A. Deutsch, L. Sui, V. Vianu, and D. Zhou. Verification of communicating data-driven web services. In *Proc. of PODS*, 2006.

[35] Alin Deutsch, Liying Sui, and Victor Vianu. Specification and verification of data-driven web services. In *PODS*, 2004.

[36] Bruce Powel Douglass. Uml statecharts. *Embedded Systems Programming*, 1999.

[37] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume B: Formal Models and Semantics*. Elsevier, 1990.

[38] K. Etessami and M. Yannakakis. Algorithmic verification of recursive probabilistic state machines. In *TACAS*, 2005.

[39] K. Etessami and M. Yannakakis. Recursive markov chains, stochastic grammars, and monotone systems of nonlinear equations. *JACM*, 56(1), 2009.

[40] J. N. Foster, T. J. Green, and V. Tannen. Annotated xml: queries and provenance. In *PODS*, 2008.

[41] J.N. Foster, T.J. Green, and V. Tannen. Annotated XML: queries and provenance. In *PODS*, 2008.

[42] C. Fritz, R. Hull, and J. Su. Automatic construction of simple artifact-based business processes. In *ICDT*, 2009.

[43] W. Gaaloul and C. Godart. Mining workflow recovery from event based logs. In *Business Process Management*, 2005.

[44] T.J. Green, G. Karvounarakis, Z. Ives, and V. Tannen. Update exchange with mappings and provenance. In *VLDB*, 2007.

[45] Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *Proc. of PODS*, 2007.

[46] D. Grigori, F. Casati, M. Castellanos, M.Sayal U .Dayal, and M. Shan. Business process intelligence. *Computers in Industry*, 53, 2004.

[47] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Comp. Programming*, 8:231–274, 1987.

[48] J.E. Hopcroft and J.D. Ullman. *Introduction To Automata Theory, Languages, And Computation*. Addison-Wesley Longman Publishing Co., Inc., 1990.

[49] J. Huang, T. Chen, A. Doan, and J. F. Naughton. On the provenance of non-answers to queries over extracted data. *Proc. VLDB*, 1, 2008.

[50] R. Hull and J. Su. Tools for composite web services: a short overview. *SIGMOD Rec.*, 34(2), 2005.

[51] Robert Ikeda, Hyunjung Park, and Jennifer Widom. Provenance for generalized map and reduce workflows. In *CIDR*, 2011.

[52] D. Janssens and G. Rozenberg. Graph grammars with node-label controlled rewriting and embedding. In *Proc. of COMPUGRAPH*, 1983.

[53] Antonín Kucera, Javier Esparza, and Richard Mayr. Model checking probabilistic pushdown automata. *Logical Methods in Computer Science*, 2(1), 2006.

[54] K. Lary and S. J. Young. The estimation of stochastic context-free grammars using the inside-outside algrithm. *Computer, Speech and Language*, 4:35–56, 1990.

[55] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems*. Springer-Verlag, 1992.

[56] Thierry Massart, Cédric Meuter, and Laurent Van Begin. On the complexity of partial order trace model checking. *Inf. Process. Lett.*, 106(3):120–126, 2008.

[57] S. P. Meyn and R. L. Tweedie. *Markov Chains and Stochastic Stability*. Springer-Verlag, 1993.

[58] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4), 1989.

[59] Martin S. Olivier, Reind P. van de Riet, and Ehud Gudes. Specifying application-level security in workflow systems. In *DEXA*, 1998.

[60] T. Pavlidis. Linear and context-free graph grammars. *J. ACM*, 19(1), 1972.

[61] D. M. Sayal, F. Casati, U. Dayal, and M. Shan. Business Process Cockpit. In *Proc. of VLDB*, 2002.

[62] R. Silva, J. Zhang, and J. G. Shanahan. Probabilistic workflow mining. In *Proc. of KDD*, 2005.

[63] Alkis Simitsis, Panos Vassiliadis, and Timos Sellis. State-space optimization of etl workflows. *IEEE Transactions on Knowledge and Data Engineering*, 17:1404–1419, 2005.

[64] Marc Spielmann. Verification of relational transducers for electronic commerce. *Journal of Computer and Systems Sciences (JCSS)*, 66, 2003.

[65] Abdullah Uz Tansel, James Clifford, Shashi Gadia, Sushil Jajodia, Arie Segev, and Richard Snodgrass, editors. *Temporal databases: theory, design, and implementation*. Benjamin-Cummings Publishing Co., Inc., 1993.

[66] W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining: a survey of issues and approaches. *Data Knowl. Eng.*, 47(2), 2003.

[67] Stijn Vansummeren and James Cheney. Recording provenance for sql queries and updates. *IEEE Data Eng. Bull.*, 30(4):29–37, 2007.

[68] Victor Vianu. Automatic verification of database-driven systems: a new frontier. In *ICDT*, pages 1–13, 2009.

[69] Marko Vrhovnik, Holger Schwarz, Oliver Suhre, Bernhard Mitschang, Volker Markl, Albert Maier, and Tobias Kraft. An approach to optimize data processing in business processes. In *VLDB*, pages 615–626, 2007.

[70] Fan Yang, Nitin Gupta, Nicholas Gerner, Xin Qi, Alan Demers, Johannes Gehrke, and Jayavel Shanmugasundaram. A unified platform for data driven web applications with automatic client-server partitioning. In *WWW*, 2007.

[71] W. Zhou, M. Sherr, T. Tao, X. Li, B. T. Loo, and Y. Mao. Efficient querying and maintenance of network provenance at internet-scale. In *SIGMOD Conference*, 2010.