A Generic solution for Warehousing Business Process Data

Fabio Casati, Malu Castellanos, Umeshwar Dayal, Norman Salazar[1]

Hewlett-Packard, Palo Alto, CA, USA

{firstname.lastname}@hp.com

Abstract

Improving business processes is critical to any corporation. Process improvement requires *analysis* as its first basic step. Process analysis has many unique challenges: i) companies execute many business processes, and devising ad hoc solutions for each of them is too costly. Hence, generic approaches must be sought; ii) the abstraction level at which processes need to be analyzed is much higher with respect to the information available in the process execution environment; iii) the rapidly increasing need of co-developing the process analysis and the process automation solution and the scale of the problem makes it hard to cope with frequent changes in the sources of process data. To address these problems, we have developed a process warehousing solution, used by HP and its customers. In this paper we describe the solution, the challenges we had to face, and the lessons we learned in implementing and deploying it.

1. INTRODUCTION AND MOTIVATION

Business process improvement has always been at the heart of any corporation's goals. In the nineties, the focus of process improvement was on *automation*: workflow and other middleware technologies were used to reduce human involvement by better systems integration and automated execution of the business logic. The total or partial automation of the process and/or of part of its steps creates the unprecedented opportunity to gain visibility on process executions. In fact, executions of process steps now leave some kind of temporary or permanent *trace* in one or more systems (databases, web sites, messages in transfer on a message broker, etc). The ability to analyze process execution information and to measure the quality, efficiency, and timeliness of process execution as well as to understand areas for improvements provides immense benefits to companies as it is key to achieving the goal of better and cheaper process execution. Recent and well-known financial crises have also prompted lawmakers to impose stringent regulatory requirements for monitoring and reporting on process executions, reinforcing the need for process analysis [4].

Furthermore, process analysis and reporting is the cornerstone of one of the fastest growing businesses in the IT sector, that of *business process outsourcing* (BPO). BPO involves delegating the execution of (part of) a business process to another company. Processes that are typical candidates for outsourcing include travel expense reimbursement, invoice payment, or employee payments, and in general all those processes that every company must deal with, but that are not part of the "core competencies". A key need in BPO is that of being able to describe and formalize Service Level Agreements (SLAs). In fact, when a company executes its own processes, it does so in a *best effort* manner. When it outsources the process to other companies, it wants to formally define the quality levels that it expects. This implies being able to define and monitor SLAs for business processes. From a provider's perspective, the need is to be able to measure and report on SLAs and to analyze process executions to meet SLAs at lower cost.

The common approach to analyzing data from transactional systems is to collect it into a data warehouse (using extract, transform, load tools) and then leverage an OLAP tool to slice and dice data along different dimensions. This is also the sensible approach for process analysis.

Process data warehousing, however, presents interesting challenges. First, developing ad hoc, process-specific solutions for warehousing and reporting on process data is not a sustainable model. The problem is even more relevant in BPO, as the provider of outsourcing services needs to support (monitor and analyze) different versions of the same process for different customers, possibly also with variations in reporting requirements for each customer. Consequently, the principal objective – and one of the main challenges -- of our work has been that of developing a general and reusable solution for process data warehousing, applicable to most or all the processes in a corporation. We have verified through experience that designing a generic solution is non-trivial but possible, as the nature of the process data and of the analysis needs share key aspects that make a common solution feasible. The solution we have developed captures the common aspects while leaving room for the all-important customer-specific and geography-specific customizations.

VLDB '07, September 23-28, 2007, Vienna, Austria.

[1] In cooperation with Ralph Gerhardt (HP Consulting), Manoj Kumar (HP India), Harry Jia (HP China), and the entire HP BPDO team. Fabio Casati is now with the University of Trento. Norman Salazar is now with UPC Barcelona.

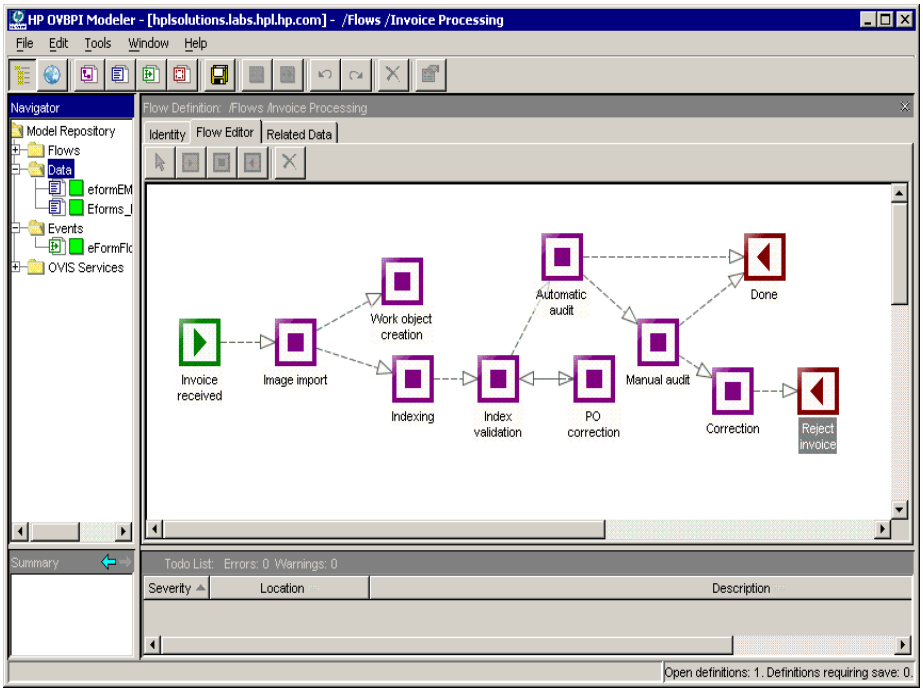

Figure 1. The invoice payment process

A second challenge in process data warehousing is that of *abstracting* process data. The typical process executed in the IT system is very detailed and consists of dozens of steps, including manual operations (e.g., scanning invoices), database transactions, and application invocations. However, reporting at this level of detail is confusing for analysts who have in mind a much higher level picture of the process. The common wisdom is that analysts, and especially business analysts and managers, perceive a process as being composed of approximately 5 to 7 steps. SLAs and key performance indicators (KPIs) are also defined on abstracted versions of a process[2].

A third interesting and novel challenge, which is today specific of the BPO domain, but which is likely to extend to in-house process execution as well, is that the business process automation application and the analysis/reporting application are co-developed. This means that the reporting solution should be in place by the time the business process application goes live, or very shortly thereafter. The challenge here is that, during development, changes to the data sources and even to the reporting requirements are fairly frequent. To make things worse, once processes are in place and start to operate in a dynamic multi-customer environment, continuous adaptations are required. Hence, it is important to devise a method for minimizing the impact of changes and be able to quickly modify and re-test the ETL (extract, transform, and load) procedures,, the warehouse model (we will use the terms "warehouse model" rather than "warehouse schema", since this is common practice in the industry), and the reports.

In this paper we describe how we have addressed these challenges as well as the lessons learned in developing and deploying our solution, including also alternative approaches that did not prove to be satisfactory. Specifically, the paper makes the following contributions:

- We analyze and classify analysis requirements for process data warehousing.
- We provide a configurable warehouse model that can satisfy complex reporting needs for virtually any process, also taking into account performance constraints. The model addresses key recurring problems such as the trade-off between the need to model heterogeneity (each process is different) and that of defining a uniform representation for all processes (to support reusability and cross-process analysis).
- We show how to abstract from low-level data about executed processes to higher-level views of the same process, suitable for reporting purposes. The approach is based on defining abstract processes and then mapping the process progression to events occurring in the source systems.
- We describe how to ETL process data, and in particular how to semi-automatically maintain ETL procedures in the wake of changes in the source applications.
- We show how the solution can be quickly prototyped using an emulation environment to get early feedback from users. This is essential, as it invariably happens that reporting requirements change considerably after users view the first version of the analysis system. Hence, tackling this problem means saving months of effort.

[2] In the following we will generally refer to SLAs and KPIs as *metrics*

2. PROBLEM DESCRIPTION AND REQUIREMENTS

2.1 Business process execution environments

This section describes, by means of a simple example, the typical process execution and source data environments on top of which the solution has to be deployed. The assumptions are very general as the solution should be applicable to different situations.

Figure 1 shows an abstracted version of an invoice management process. The actual implemented process is much more complex, and its definition may not even fit on a page. In large companies, this process is run several thousands of times per month. Due to the volumes, and to the penalties involved in late payments, it is imperative that it be performed in an efficient and timely manner. Hence it is one of the many processes for which analysis is important.

The process begins with the receipt and scanning of the invoice document in case of paper invoices. Then the data is indexed (extracted from the image and entered in a database) and validated. Then, also based on comparison of invoices with purchase orders, the order information as well as vendor payment information are updated. The invoice and the information updates are then audited (via automatic error checks, and possibly a manual audit which is performed occasionally depending on the seniority of the employee who has done the initial validation). If the invoice passes all the checks, then the process ends (the invoice will then go through payment, not described here). Otherwise, it will be returned to the vendor.

Processes are typically supported by a combination of systems and technologies, which include scanning and document management systems, databases, ERP systems, custom applications (including Web applications for data entry or to facilitate/track approvals), and in some cases also workflow systems which automate part (but rarely all) of the process logic. In general, information from all these systems is needed for process analysis and reporting.

2.2 Common reporting requirements

Over the past couple of years we have analyzed reporting requirements for many different processes, with the goal of identifying common requirements. This makes possible a unified approach to reporting and analysis, so that the effort for setting up the analysis for a new process or customer consists mostly of customization rather than development. We have found that all of the reporting requirements can be classified along the following categories of metrics:

- *Process metrics*: these are based on process progression data only (i.e., activation and completion of steps), and include:
  - Metrics on basic process statistics (process and step durations, or volumes)
  - Metrics on the time interval between the start/completion of a step and the start/completion of another. For example, we may want to monitor the time taken from *indexing* to *correction*, because this is a measure of the performance of a certain department.
  - Path and outcome metrics, such as the number of times a loop is executed (e.g., how many times a correction cycle is needed) or the percentage of time a process ends at a certain end node.
  - Correlation with previous step: very often we have encountered the need to report on steps in relation to the previous one. This is common when there are some exception-handling steps which are executed when the previous step fails. Note that the "previous step" has to be determined dynamically, since in general different instances can take different paths through the process .

  -- *Resource metrics*:
  - Performance of human and automated resources in executing steps.
  - Correlation between resources and process metrics (which resources statistically lead to successful or unsuccessful executions, or which resources have statistically led processes to follow certain paths in their execution)
- *Business data metrics*:
  - Correlation of *business data* (e.g., invoice data, vendor data) with process data. For example, analysis of efficiency and quality of execution based on invoice type.
  - Correlation between business data and resources. For example, number of invoices from a given center processed by a given employee.

The underlying requirement for all of the above metrics is that they are defined and computed on abstracted versions of the process, not on the actual implemented version. We have witnessed this need consistently throughout all reporting requirements, except when the actual process was itself very simple.

In the following we show how we have addressed the requirements on the metrics via a data warehousing approach that is process-independent, and how we have addressed the requirements on abstraction via a mapping and correlation mechanism that drives a process-aware ETL procedure. We then show how we have tackled the problems related to application-reporting co-development and rapid prototyping.

3. PROCESS DATA WAREHOUSE MODEL

Defining a generic model for a process data warehouse has the following challenges:

- Multi-level instance data (facts): a process execution has related *facts* at different levels of granularity, including step-level facts (e.g., step durations), process instance-level facts, and data-related facts (values and changes to business data). Facts may have to be self-correlated, especially in relation to the need of reporting on the "previous step".
- Business data associated with a process instance is always different from process to process, can in general have complex structures, and can change at every step

during the process, so that its representation becomes complex (and hard to generalize).

– Process and step executions go through a lifecycle. For example, steps are created (they are ready for execution), assigned to users, locked or activated (people begin to work on them), reassigned to different users, unlocked (people stop working on them), and completed. All these are events that should in principle be modeled. They are again facts, which are at a granularity even lower than that of the step (i.e., at the level of step status changes), and which depend on the business process and on the system supporting them (e.g., different workflow systems have different lifecycle phases for steps and processes). Also, the number of states in the lifecycle visited for each step execution is in principle unlimited (for example a step can be suspended and re-activated several times).

Figure 2 shows the most significant elements of the data warehouse model. Representing dimensions is fairly easy and, since it does not pose any particular new challenges, it won't be addressed here. The key challenge is how to represent facts. At the *step-level*, the approach we took is to model all step execution facts in a single table, with one row per step execution (see table *task execution* in Figure 2). This design has some limitations: one is that we represent information from different source systems (and for processes that have different step properties and lifecycles) within the same data structure. The other is that we "compress" the possibly infinite set of step-related events into a limited set of attributes. However, we have experienced that this approach is more effective than the alternatives of i) representing step facts at the granularity of step status changes and ii) using different step tables for information coming from different systems, and hence having different properties.

In fact, the vast majority of step-related queries are related to a limited number of measures (namely, step durations, total time in which a step is "idle", or suspended), which can be computed at ETL time and which can be summarized at the step execution level (see duration related columns, e.g., *New_To_Ended_Duration_Sec*). Furthermore, in terms of analyzing the resources that performed the steps, what is always queried is information about the first user to whom a step is initially assigned for execution, and the last user who finally completes the step (see processor-related columns, e.g., *Final_Proc_Key* – note that in business process terminology, users who execute steps of a process are called "processors"). Though steps may be re-assigned from user to user, this is infrequent and it has never been a target of reporting requirements in any of the cases we have analyzed. The issue of sourcing data from processes in which step lifecycles follow different models is solved by recognizing that in practice there are only three step states that are relevant: *creation*, *activation* (somebody begins to work on a step), and *completion*. This is sufficient to model reporting requirements in virtually any system, and more complexity than this is unnecessary as it is not understood by business users. Finally, modeling tasks in one table is very convenient as it allows us to perform analyses that go across processes (e.g., analyze resource performance or aggregating measures on step–level SLA violations regardless of the process). Performance is not impacted as bitmap indexes and partitions make it feasible to deal with high volume tables when process-specific analysis is needed.

The problem of self-correlation is solved by "unfolding" the correlation in the tuples. While we often found that queries ask for information on a step with respect to the previous one (e.g., retrieve all the steps that lead to the subsequent execution of the *index validation* step), the kind of information required on the previous step is limited. Hence, we add columns (see those with prefix *Previous*) to the *task execution* fact table to explicitly store this information, computed at ETL time. The alternative option was to store in each tuple a link to the entry in the same table that describes the previous task. However, this leads to much slower queries (hundreds of times slower) unless we create a unique index for fast access to step data, but with considerable space penalty and data loading performance issues.

At the *process instance* level, a different approach is required. Here, we still define a generic process instance fact table that contains basic process information (e.g., start time, end time, duration, initiator, etc). However, we need to store business data as well for the analysis. While in general business data can have arbitrary structures, the typical need is that of representing a (process type-specific) tuple of data elements for each instance, such as <invoiceID, customerName, Amount,..>. Hence, the solution here is to create a table per process type, which contains process specific data (e.g., Invoice_Business_Data in Figure 2) along with process instance-level progression data, essentially replicating much of the information present in the *process_instance* table. For example, the Invoice_Business_Data table has the Receive_Date_Key element that represent the date in which the process instance starts. Similarly, the payment date corresponds to when the instance ends.

This replication is necessary to simplify correlated business/progression analysis, as joining with the *process_instance* table should be avoided if possible. Note that the *process_instance* table is still needed as often it is necessary to perform analyses that go across instances of different processes. The size of the tuples is typically quite small so this duplication is feasible in the vast majority of cases.

Perhaps surprisingly, queries that require correlation of step-level information with process instance or business data are infrequent. However, they do occur, and, although this is discouraged by many, we handle this via links (referencing columns) between the fact tables at the step and process granularities (from a step fact to its process instance fact, and from the process instance fact to its business data fact). In Figure 2, for example, the Top_Work_Object_Key in the task execution data refers to the process instance data, while the Entity_Table_Key in the process data refers to the business data. We found that performance is acceptable (with proper indexing) in most cases, unless we go into volumes in the order of several tens of million of rows.

Note that we only have one generic step instance table but many business data tables; hence the pointer to the business data table (Entity_Table_Key in Figure 2) is *blind*: it does reference an entry in *some* business data table, but which one is not specified. This is not a limitation as when

such a step-process correlation is needed along with (process type-specific) business data, the analysis is necessarily process type-specific, and the query writer has to be aware of which business data table to join. This approach does not preclude more complex structures for business data (e.g., having an invoice line item table along with invoice data), as long as there is a "master" business data table for each process.

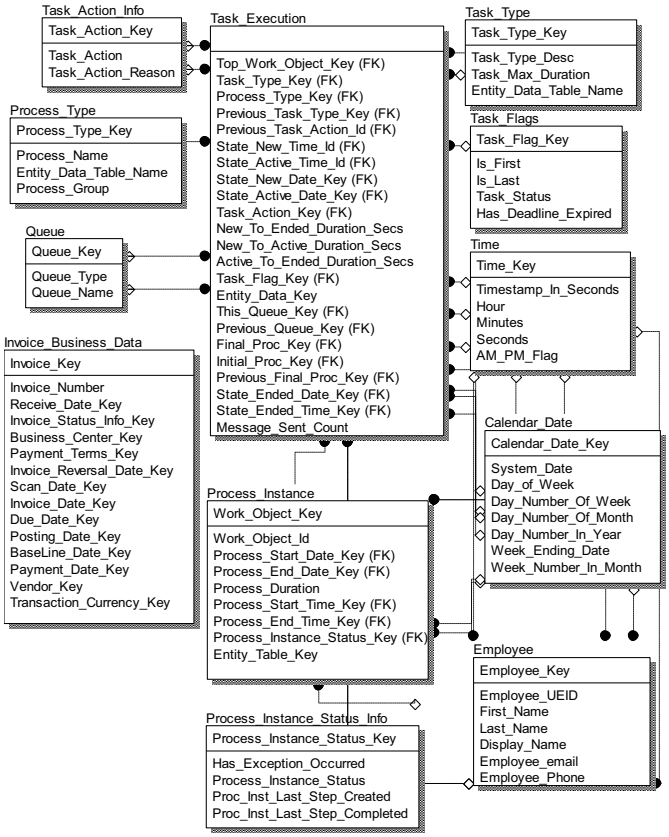

Figure 2. Process warehouse model (not all attributes are shown)

In summary, the key solutions are: i) single granularity for steps; ii) single fact table for any step of any process, with aggregation of most common measures that are at the step event granularity; iii) correlation with previous step data handled via additional columns; iv) separate business data tables per each process type; v) blind links to handle step-process correlation with business data. This structure works for any process, and can be easily implemented.

These design decisions are the result of a large number of experiments and the exploration of many different alternatives. Having this common design shortened the warehouse design cycle for new processes to nearly zero, and report writing has also been considerably simplified due to the many commonalities in the reports for different processes. This translated into very significant savings in development time. Indeed, the first design took about six months, while the addition of a second process only required three weeks. Note that these savings occur both when we add a process over the same schema as well as when we create a new, separate schema, but use the above as "guidelines" for process data warehouse design.

4. COLLECTING, ABSTRACTING, AND LOADING PROCESS DATA

4.1 Mapping events to abstract processes

The problem of providing abstracted process representations for warehousing and reporting has two facets: first, we need to provide users with a way to *model* the abstraction, that is, to describe the high level process and how its progression maps to underlying IT events. Second, we need to have an ETL mechanism that, based on the abstract process definition and the events occurring on the different systems, loads the warehouse with abstracted process execution data. Modeling abstract processes involves the following:

1. Describing the process flow (Figure 1) as needed for analysis, along with relevant business data (also possibly abstracted with respect to the actual business data).
2. Specifying how the abstracted business data for each process is populated and maintained, based on a mapping between business data and IT events that mark the progression of the process. For example, one can define an *invoice* business data structure, and state that the *audit result* attribute should be populated when a message of type *audit* is sent over a message bus, and the value of parameter *response* in this message should be taken as the value. Note that the mapping must include a definition of the *correlation logic*, information used to associate events to the correct business data instance (and indirectly to the correct process instance, see below). For example, one can specify that, when the *audit* message occurs, only instances of invoice whose *invoiceID* attribute is equal to the *invoice number* parameter of the message should be updated (Figure 3).
3. Associating the start and completion of each step with changes to the abstract business data, to define progression information. For example, we associate the completion of the audit step with the fact that the *audit result* attribute becomes populated (Figure 3). In this way, we can map IT events, through the business data, into events at the level of the abstract process instance.

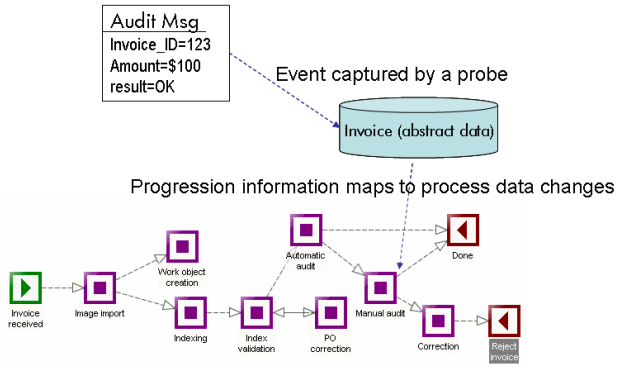

Figure 3. Mapping audit message event to audit step completion

4. Associating steps to human or automated resources. This is again based on mapping to the abstract business data. For example, one can specify that the resource that completed the audit step is to be determined by looking at column *auditor* of the invoice abstract data.

Modeling is performed via a component called HP Business Process Insight (BPI) [2]. This is a stand-alone tool that can also be applied as a real-time process monitoring solution (the attentive reader will have noticed that if we are able to capture events in real time, then we can "enact" the progression of abstract processes in real time), though here we describe its concepts and application in the context of process data warehousing.

BPI provides a Java interface that allows users to provide the specifications described above (Figure 1). Specifications are then accessed by the ETL to know how to map process data across levels of abstractions, and how to construct business data (Figure 4). Event data is captured by deploying *probes* that monitor source systems for events of interest and, when the event occurs, store the event occurrence information (event name and its parameters) into an event log database (one table per event, storing event name, occurrence time, and its parameters, one per column). For example, a table *audit* that stores audit events would have attributes *eventID, timestamp, invoiceID, result, auditor*, etc. The probing mechanism used (*openadaptor* [3]) is provided by a third party and is not part of our work.

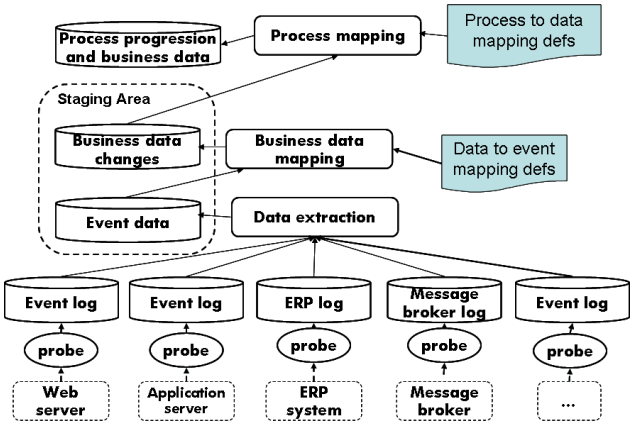

Figure 4. Extraction and abstraction of process data

At data loading time, the ETL function goes through the events (reads all the event tables, whose names and types are also stored by the modeler, so they are known) orders them by time, and, based on the defined mappings between data and events, it generates a set of changes to abstract business data, i.e., a change log, timestamped, based on the event timestamp. For example, it creates a set of records for the invoice abstract process data, possibly with many records for the same invoice instance, if more than one event for that invoice occurred in the extraction period.

Next, the process progression is computed, based on the mapping between process and data changes, by "replaying" the data changes in the specified order and detecting changes that are relevant for the progression. At this time, besides marking the progression (i.e., creating records for the step execution data), the latest value of the abstract business data is also stored, and this information is then ready to be loaded in the warehouse.

An alternative modeling approach involved mapping the process progression directly to events instead of indirectly, via changes to the abstract business data. However, this level of indirection is helpful given that i) many different events may cause the same change to a business data item (e.g., if the process is implemented on a different IT infrastructure), ii) the same business data can sometimes be used to support and mark the progression of instances of different process types, and iii) in practice, for abstract processes the progression is often dependent on business data changes, and in the rare cases where this is not true, an ad hoc dummy data element can be created in the business data for this purpose. Hence, this approach on average reduces the specification and maintenance effort and makes the specifications more robust to changes in the information sources (which often requires event specifications to be updated, but no changes are needed to business data or progression information).

4.2 ETL generation and maintenance

Analogously to what we did for the process data warehouse model, we have been able to factor the common aspects of ETL into a generic solution that applies to any business process. In contrast to the way ETL is normally done in data warehousing, where data extracted from the source(s) lands into a staging area and a single transformation stage maps it into the target warehouse tables, our solution requires a *two-phased* transformation stage (see Figure 5). This two-phased transformation is the result of our abstraction requirement where the events monitored in the underlying systems are mapped to data changes -first phase of the transformation-, and from those to process progression –second phase of the transformation- (see Figure 4). The mappings used in both transformation phases are those defined with BPI as part of the abstraction modeling (see #2, #3 in section 4.1) The main challenges that we faced were to find a way to semi-automate the creation and maintenance of the staging area, and the execution and maintenance of the mappings. To this end, we devised two generic procedures as part of our solution:

- One procedure automates the creation and maintenance of the *staging area* where event data extracted from the source logs not only lands but is mapped -- using the mapping procedure described below -- to a set of changes on abstract business data (first phase of the transformation).
- The other procedure automates the generation of executable transformation scripts from the mappings specified in the abstract process model (section 4.1). Such transformation scripts are used for mapping data to different abstraction levels in each transformation phase.

One important design principle was that of being agnostic with respect to the underlying tool (i.e., home-grown or commercial) supporting the execution of the ETL procedures automatically generated by our solution.

4.3 Staging Area

The staging area in ETL is a database (could be files but we have chosen to use a database) where extracted data is

staged to get it ready to be loaded into a data warehouse. It basically serves two purposes: as a landing area where extracted data lands, eliminating the need to repeat an extraction if anything goes wrong (extracting data may impact the operation of the source), and as a working area where data is checked and prepared for loading. It is in the staging area where different sets of tables need to be created for various purposes. Normally, landing and image tables (explained below) are created, but our solution also includes an intermediate set of tables to handle the two-phased transformation stage where the result from the first transformation phase needs to be staged to be used as input to the second transformation phase.

To populate the process data warehouse it is necessary to first extract the data from the different event log databases into the *landing* tables of the staging area. To define the schema of such tables we use a GUI that allows the user to check off the tables and fields from where data will be extracted from each event log. This procedure generates a schema definition script in SQL DDL for creating the corresponding landing tables.. For example, if the message broker log had a table for audit messages with fields <*invoice_id, amount, result*> and all of these fields need to be extracted to map an audit message event into a change to the *result* data of the corresponding *invoice*, then they will be checked off and the following DDL statement will automatically be generated:

USE LANDING_AREA

CREATE AUDIT TABLE (invoice_id char(15), amount decimal (6,2), result char{12))

Normally, once data has been extracted into landing tables, it is checked for duplicates and for determining if it is an insertion, update or deletion. To this end, the tuples in the landing tables are compared with their counterparts in the *image* tables. Image tables, as the name suggests, keep an image of the last version of the records extracted from the sources since the first extraction cycle (or since the last time the staging area was flushed). There is one image table for each landing table with exactly the same schema. Therefore, the same DDL script used to create the landing tables is used to create the image tables as well. Once data in the landing tables has been checked, erroneous data is sent to error tables for later reprocessing, and non erroneous data is copied to the image tables, while the landing tables are truncated just before the next extraction. Scripts are generated to do the necessary comparisons and detection of errors. For example, for each landing-image table pair, a script is automatically created to compare the business key of each tuple in the landing table with those in the image table. If a match is found, the remainder of both tuples is compared, and if they still match, then the tuple in the landing table is discarded as it is considered a duplicate (i.e., a tuple that had already been extracted in a previous ETL cycle). If the rest of the tuple doesn't match, then the tuple in the landing table is marked as an 'update' so that once the tuple is transformed, it is treated as an update to the corresponding tuple in the warehouse. If there is no match on the business key of a tuple in a landing table, then it is a new tuple and therefore it is marked as an 'insert'. For example, if there is a tuple <100, 500, 'reject'> in the audit landing table with schema <*invoice_id, amount, result*>, and in the audit image table with same schema another tuple with invoice_id=100 is found, and the values of its other attributes are <500, 'reject'>, it means that the tuple had already been extracted and processed in some previous ETL cycle. When a tuple is either a new tuple (i.e., insert) or an update, it is timestamped. It is the new data in the image tables that is transformed into the target tables of the process data warehouse.

To implement the two-phased transformation, we have introduced another set of tables, which we call *intermediate* tables whose purpose is to stage the output of the first transformation phase to be used as input for the second transformation phase. These tables have the same schema as their counterpart business data tables in the process data warehouse. Therefore, the same DDL used to create those tables, is used for the creation of the intermediate ones. In contrast to the other two sets of tables (i.e., landing and image) which are populated by an SQL statement of the form INSERT INTO *table* SELECT *attributes* FROM *source_table*|*landing_table*, the intermediate tables are populated by the execution of the mappings from IT events to business data changes (#2 in section 4.1) specified as part of the abstracted process model defined for BPI. For example, once the audit message has been extracted into the corresponding *audit* landing table, and it has been detected and marked as a new (i.e., insert) tuple and copied into the *audit* image table, the tuple is mapped into the progression of the audit step marking its completion according to the mapping that associates the completion of the audit step with the fact that the *audit result* attribute becomes populated. In the next section we will describe how the mappings are executed via scripts that are automatically generated.

An alternative approach could be to have two separate ETL processes, one that extracts data from the event sources and maps it into business data changes loaded in the process data warehouse, and another that extracts business data from the process data warehouse and maps it into process progression data loaded into the warehouse. However, this alternative is less efficient given that business data changes need to be extracted (after being loaded into the data warehouse) and staged to isolate the warehouse from any impact when extracting such data for the second ETL process. Furthermore, additional tables would have to be created in the warehouse because the business data tables in the process data warehouse only keep one record for each business data instance (e.g., per invoice). Thus,,all the change records, but the last one, for a given business data instance coming from the same extraction would be lost, but these would have been needed to mark the corresponding progression of the abstract process through different process steps.

Once the structures of the different tables in the staging area are created, they need to be populated. Landing tables are completely refreshed at every extraction cycle by simply inserting the data extracted from the event log sources into the corresponding landing tables using INSERT-SELECT SQL statements. Image tables are incrementally refreshed at every extraction cycle by copying the appropriate tuples (i.e., new inserts and updates) from landing tables into the corresponding image tables. Intermediate tables are

populated as the result of executing the mappings from IT events to business data changes. Finally, data in the intermediate tables is mapped to process progression data loaded into the target tables of the process data warehouse. The procedure to automatically execute both kinds of mappings will be explained in the next section.

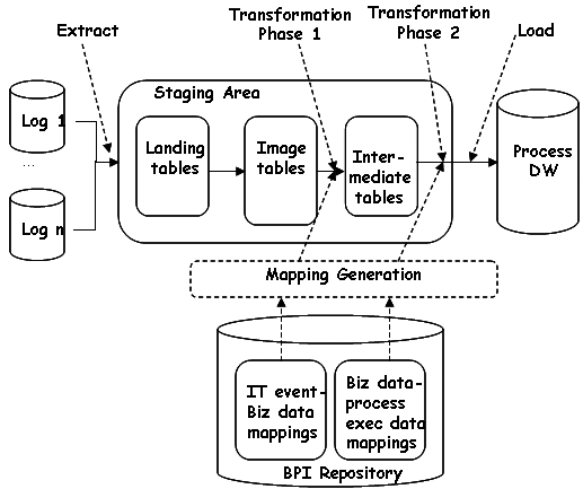

Figure 5. Transformation phases

A solution that creates and populates the staging area is not complete if it cannot cope with change. Log structures and business data structures may change, so it is important for the staging area to be automatically maintained. Our solution can detect changes to the source schema by periodically retrieving and comparing the source schema definitions with their previous versions, prompting the user to identify those changes that are important from a reporting perspective (e.g., not all columns of a newly added table may be required for warehousing and reporting, whereas modifications or deletions of columns with a counterpart in the staging area always have an impact). For those changes identified by the user as relevant, corresponding ALTER TABLE statements to modify the staging area schema are automatically issued.

5. TESTING AND RAPID PROTOTYPING VIA EMULATION

We mentioned earlier in the paper that one of the main issues with process data warehousing, especially when many different processes are analyzed and even more so in a BPO environment, is that of co-developing the warehousing solution along with the process management applications. Such co-development has two major issues: i) unlike traditional warehousing problems, source data is typically unavailable until very late in the project, as the source applications have not been developed yet; ii) the source and their data stores change frequently as redesigns are made during the development cycle. The implications are that it is hard even to begin development test of the ETL, warehouse, and reports, and waiting until the completion of the source application easily implies delays of several months.

Even more importantly, we have experienced that customers and analysts consistently give the wrong reporting requirements in the beginning, because of poor initial understanding of the process and of the capabilities of data warehouses and reporting tools. Hence, it is absolutely essential to rapidly prototype the warehousing solution (well before the source process application has been completed) to get feedback on the reporting requirements and incorporate the many requests for change that invariably come. Though this may seem like a minor detail, it is actually a major headache for any reporting solution and can easily result in millions of dollars in extra development costs and delays.

To address the problem, we first recognized that what we needed was an emulation environment that supports testing and prototyping of:

- Events and data in the sources generated according to the correct process logic (the order of steps in the process), and corresponding to events and data useful to mark the progression of the flow.
- Data on resources that contribute to the step executions, e.g., on users and applications that contribute to executing steps. This data should be correctly correlated, meaning that delays in application executions should correspond to delay in step executions. This consistency is important again for the purpose of receiving user feedback, but it helps considerably also to detect errors in the ETL procedure.
- Business data associated with the process. Again, this should be as meaningful as possible to collect feedback. Completely random data, such as randomly generated strings to denote customer names or meaningless invoice amounts, make it very hard to get valuable feedback.

To perform validation and prototyping, we need to generate the above in a way that is as realistic as possible. This means that we actually need to fill the sources with process data and business data, we actually need to deploy probes to collect events and load event logs, and we actually need probes to monitor resources and collect resource data. The importance of *emulating* rather than *simulating* the data is that now we can actually test the data extraction process, including those based on probes that monitor source systems and generate monitoring data or extract business data.

We also need flexibility to simulate different conditions (resource unavailability, poor performances, SLA violations, and the like) both for the sake of testing and for user validation. All of the above can only be achieved by *emulating* the process-based application.

We next observed that, in process-based applications, there are two specifications that are available early and that are reasonably stable: i) the process model, and ii) most of the database schemas. They are the first artifact being designed and where the most care is placed. Furthermore, if specific tools (e.g., workflow tools) are used for different parts of the process, then the schemas of the databases and logs written by these tools are known (and stable).

Once we have process models and data schemas, we can emulate the execution of the process. We created an infrastructure for this. The infrastructure includes a process execution engine (PEE), used to model the process, which can be the actual one or a simplified version of it, but

sufficient at least to match the level of detail of the desired abstract process. For example, we can model the process of Figure 1. We currently use Weblogic as PEE, but any other modern workflow system would serve the same purpose.

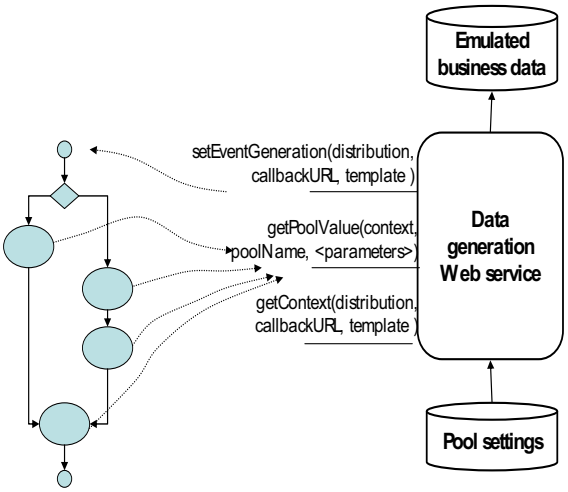

Figure 6. Process emulation via the data generation web service

Each step in the process is assigned for execution to a *data generation* web service. This is a Web service that we developed and that has three main purposes: i) wait a random amount of time before returning from a method call (the random function can be specified), to emulate executions of different durations; ii) generate meaningful business data, e.g., customer names; iii) act as a configurable event generator, which produces events of different types, with parameter values following different distributions, and with event generated according to random functions. Such a service can provide for all the process enactment needs: it can generate events that create new process instances, can simulate step executions of different durations when invoked by the steps, and can return meaningful business data (Figure 6).

The data generator service has a number of built-in pools with, for example, names of (fictitious) people, email addresses, names of customers, and the like. Clients can request an entry from the pool, resulting in a name being returned, chosen randomly. Alternatively, names in the pool can be ordered so that certain names are returned more often than others, based on probability distributions that are part of the pool definition. Numeric pools are again based on probability distributions, or can be sequences of values, with the meaning that each time a sequence pool is accessed, the next value in the pool is returned. Events of interest can be defined by registering a template (setEventGenerator function in Figure 6), which is an XML document that may contain parameters, represented by pool names. Based on the time distribution specified as part of the function call, an event is generated at different point in time. The event is represented by the XML document where pool names are replaced with pool values, again taken on the distributions which are part of the pool definitions.

Once a process is started, each step binds to the data generator service and calls the *getPoolValue* function, which requests a value from a specified pool to be returned, after a random *duration* has elapsed (Figure 6). This return value is used to edit process data in the PEE and drive the execution of the process (e.g., take a certain path based on decision conditions).

The call also includes the possibility of defining a SQL statement, which is parametric analogously to the event template defined above. The SQL statements describe which tables should be modified by the service execution, and how. It is executed at the end of the waiting period. This allows for the generation of source business data. For example, in the audit step, we could include the following statement in the call:

```
INSERT INTO AUDIT_RESULTS (INVOICE_ID, AUDITOR, RESULT) VALUES (%INVOICE_ID%, %PERSON%,%YES-NO-RESULT%)
```

Though not detailed in this paper, calls to the data generator web service occur within a *context*, which can be created for each process instance and that is useful for example to specify that all invoice IDs generated for the same context have the same value. The context can also be used to specify DB connection information, so that the web service knows where to execute the SQL statements.

This infrastructure is the essence of what is needed to emulate the generation of meaningful source data, which in turn is the basis for both testing ETL and warehouse as well as for quickly building reports with meaningful data to get feedback from the users. The process described above can be developed quickly, as we do not have to implement the steps (the implementation is represented by the data generation services), and specifying the flow logic is also easy, especially if at first we are interested in rapid prototyping more than testing, and hence we can emulate the abstracted process rather than the actual one. It does not provide all the flexibility that may be required in the general case (e.g., we may want to simulate a behavior in which invoice audits are successful based on invoice amount or based on the person processing the invoice, and this requires custom coding), but we have experienced that this is sufficient in most situations.

6. DISCUSSION AND CONCLUSION

Our work significantly extends our previous contribution on the subject [1], which was limited to warehousing data coming from workflow systems (and hence with fixed, known schema) and did not deal with abstraction, modeling of reporting requirements such as step correlation or step status changes, ETL maintenance, and rapid report prototyping. We also introduce significant variations to the original model design based on accumulated experience.

The other closely related work is that of workflow analysis systems and business activity monitoring systems, such as [5,8]. These efforts provide a warehouse schema that is independent of the business process, but that is dependent on the process meta-model, as that is built into the workflow engine. There is no specific capability to collect and aggregate data coming from sources that are not the workflow engine itself, neither in terms of warehouse model

or ETL, and there is no support for process abstraction. Similarly, there is no support for rapid prototyping.

During the last two decades there has been a vast amount of work for automating the generation of mappings. One of the best efforts is the Clio project from IBM [6] where given value correspondences between source and target schemas, Clio can produce logical assertions, basic and nested [7], that can be converted into executable programs. Our work differs from those efforts in that our solution doesn't produce logical mappings that exclusively match the users specified correspondences, instead it produces mappings that capture other correspondences not specified by the user but that are part of the execution semantics of abstract process progression by factoring out commonalities derived from the predefined structure and semantics of the process warehouse model and of the types of mappings specific to the process warehousing domain. To the best of our knowledge, we are not aware of any work done in this direction.

Finally, we mention that the framework described above has been validated over a variety of HP and customer processes. Although it can be implemented with a variety of databases, ETL tools, and reporting tools, HP's implementation is based on an Oracle database for the warehouse, BusinessObjects' Data Integrator as ETL, and BusinessObjects XI for the front end reporting.

While we did execute performance tests for high volume data and the model and approach is targeted for high volumes (especially as in BPO environments there are many customers on the same platform and high volumes are the norm), to date our customer scenarios only included applications with relatively low volumes of transactions, in the order of tens of thousands per month with overall sizes of fact tables in the order of a few millions. Another limitation is that the abstraction aspect only applies when it is reasonably easy to associate process progression with IT events. This is true in the vast majority of the cases, though in some situations the mapping is not trivial, mostly due to the difficulty of correlating events with the correct process instances. For example, the start of a step may be associated to a user accessing a web page (say to enter invoice information), but this event per se may carry no additional contextual data and hence although we know *indexing* (i.e., the process step corresponding to invoice data population) is being done, we do not know for which process instance. Associating the event to the correct process instance may require additional parameters such invoice number or customer id.

REFERENCES

[1] Warehousing workflow data: challenges and opportunities. Procs of VLDB'01. Rome, Italy. 2001

[2] Business Process Insight 2.0. Information available at www.managementsoftware.hp.com/products/bpi/

[3] Openadaptor Programmer's guide. Nov 2005. https://openadaptor.openadaptor.org/pg/

[4] www.sarbanes-oxley-forum.com

[5] FileNet Business Activity Monitor. Available at filenet.com/English/Products/All_Products/bam.asp

[6] Schema Mapping as Query Discovery. R.J.Miller et al. Proceeding of VLDB 2000.

[7] Nested Mappings: Schema mapping reloaded. A.Fuxman et al. Proceeding of VLDB 2006.

[8] WebMethods. Business Activity Monitoring: The new face of BPM. June 2006.