The Business Data Object Versioning and Change History Patterns

RUSS RUBIS, Florida Atlantic University
IONUT CARDEI, Florida Atlantic University

Businesses constantly consume, generate and share data. Without data, a typical business process would come to a halt. But for the data to be useful to businesses, it must be accessible, manageable, current, and it must be traceable. Because enterprise applications are designed for multi user environments, the audit trail of any data creation, update, and deletion is of paramount importance. Users (and processes) should not be allowed to change business data without some sort of audit trail linked to the said change. Furthermore, often changes to certain types of data require approvals, validations and verifications, which often take time and are not instantaneous. Thus the requested changes cannot be performed directly on the original set of data, but must rather be done on a data object that is the exact copy of the original. This approach allows for the existing data object to continue to be available within a system while the requested changes are being approved, validated and verified. Furthermore, should the requested changes be rejected or denied, the original data stays intact. In this paper we describe several patterns on Business Data Object Versioning and Change History that are useful for designing enterprise applications in need of adherence to legal and corporate audit record creation and maintenance.

General Terms: enterprise applications

Additional Key Words and Phrases: business data object, change history, change tracking, audit trail, change traceability

1. INTRODUCTION

In typical day-to-day operations of almost any enterprise, the front-end users of the enterprise applications are mainly concerned with completing their assigned tasks in the most efficient way possible. But in a multi-user environments, it is often beneficial, and at times required to keep track of who made changes and when, and even what changes were performed. The granularity of the required change tracking is often driven by the type of business, or the type of data being changed, or even economic, political, and legal forces outside of the business itself. For these reasons, there is no "one glove fits all" approach when it comes to designing and implementing audit trail solutions. In this paper we present three design patterns addressing the tracking of changes to business data within a multi-user enterprise application. The emphasis of the patterns presented in this paper are on addressing regulatory requirements of a business, rather than branching and merging patterns, which have already been sufficiently covered in numerous literature. The goal of the following patterns is to address the issues of audit trail, such as who updated a business data object and when. The issues of session management and mutual exclusion of updates, which are essential, are out of scope of this paper. Often, businesses require just a minimalistic approach to tracking changes for most of their data. The first pattern in this paper addresses this common scenario by proposing a record keeping mechanism of who made changes to a business data object, and when the changes were made. There are times however, when knowing simply who made the change and when is not enough to maintain a proper audit trail from business and/or legal perspective. Scrupulous accounting practices, and in particular manipulation of accounting data, at Enron in the late 1990s for example, led not only to the collapse of the company, but also to changes in the regulatory requirements, resulting in the introduction and passage in the U.S. Congress of the Sarbanes–Oxley Act of 2002 [1] The next two patterns offer an extended approach to tracking changes to a business data object by introducing a versioning pattern for preserving historical data of a business data object (second pattern), and the concept of change history records which are maintained outside of the business data object itself (third pattern). Managing granular access to fields and attributes within a given business data object is a topic in itself, and has been covered in the "Patterns for fine-grain access-controlled business objects" [11].

2. THE SIMPLE CHANGE HISTORY PATTERN

2.1 Intent

In a multi-user environment changes to business data can be frequent and performed by more than one user. Knowing who was the last person to perform a change and when can be beneficial for audit trail and accountability. In this pattern we introduce the concept of simple change history, which enforces recording of the owner and date of the last change performed on a business data object.

2.2 Example

A Human Resources (HR) application maintains a record of employees, including their personal and salary information. A new employee has been entered into the system by an HR clerk who entered the new employee's salary incorrectly. An HR manager, realizing the mistake then updated the new employee's salary to the correct amount. An audit trail for the initial creation and the subsequent update exists, recoding the owner, and the date and time of the initial employee record creation, as well as the owner, and the date and time of the subsequent update.

2.3 Context

You are designing a multi-user business application in which a given data object can be created and subsequently updated by different users at different times. Only the most recent update needs to be recorded.

2.4 Problem

How to keep track of who created or modified a given business data object and when.

2.5 Forces

- It is important to know which user created/updated a given business data objects and when.
- Insufficient or vague audit trail requirements.
- Implementing audit trail in legacy systems where ability to change existing data structure is very limited.

2.6 Solution

"A business data object is a software artifact often used by business applications and services and it is a widely acceptable entity in the running of the business." [13] In above example an employee record is considered a business data object as it is part of an HR application and has a well-established and commonly accepted attributes and behavior. One of the simplest ways to track changes performed on a business data object is to store information on the most recent owner and the date (and time) of the change, as shown in Figure 1.

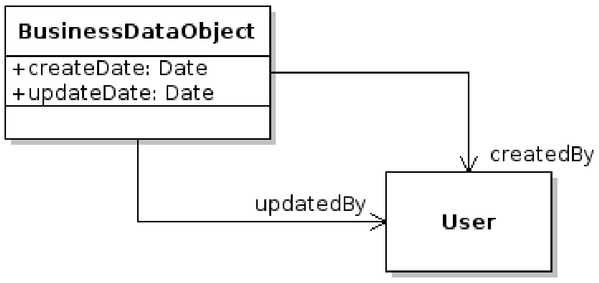

Figure 1 UML class diagram for the BusinessDataObject class, with change history attributes and associations to a User class.

The *User* class represents the identity of the user or process managing the object. If the object is being updated or created by some process within an application, then the User class would represent the owner of that process. By adding the *createDate* and *updateDate* attributes, and the *createdBy* and *updatedBy* associations to every business data object, we can enable a simple change history tracking of each data object. Every time a user or a process performs a change to an existing business data object, or creates a new data object, we can track the owner and date of the change. When a data object is initially created, the *createdBy* and *createDate* attributes are set to the user object that initiated the creation of the said object, as well as the date the creation took place, respectively. Consequently, when updates occur to the given data object, the *updatedBy* reference and the *updateDate* attribute are assigned to the corresponding user object and the date of the update, respectively.

2.7 Consequences

The Simple Change History Pattern enables an application to store the information on who performed the creation or update of a given business object, and when, but the pattern falls short of keeping track of multiple updates to a given business data object. Because there is only one attribute for update owner and one for update date, the pattern allows for recording of the most recent update information only, thus resulting in the loss of the previous update records. The pattern is a good fit when requirements are not detailed or specific, or dealing with legacy applications, where significant changes to business data object might not be feasible, or when requirements for audit trail do not call for elaborate or complex approaches.

2.8 Known uses

Most major Commercial Off The Shelf (COTS) Enterprise Resource Planning (ERP) developers such as Oracle and SAP, to name the two biggest, implement this pattern. The owner and date of change attributes are added to almost every business data object in these types of applications. In particular, Oracle Applications have many tables which have CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, and LAST_UPDATE_DATE columns on most of their master data tables, such as PO_HEADERS, PO_REQUISITION_HEADERS, PO_VENDORS, etc. [12]. These columns are used to store the owner and date of creation and change of every row of data within those tables.

3. THE BUSINESS DATA OBJECT VERSIONING PATTERN

3.1 Intent

There are often times when a simple record keeping of who made the changes and when is not sufficient because changes to a business data object are complex, may take time, and/or require review, validation and verification. At the same time, the business data object still needs to be available and accessible within the given application. The Business Data Object Versioning Pattern provides a solution by introducing a new version of a given business data object which allows preserving the original data, and continuous validations and approvals of the proposed changes to a given business data object, without making them available to the application until all approvals and validations are completed.

3.2 Example

A purchase order has been issued to a vendor for the purchase of two new laptops. However, soon after the purchase order had been sent to the vendor, it has been determined that the *ship to* address was entered incorrectly and an additional laptop is required. A change order (or a new version) of the existing order with the correct ship to address and the updated laptop quantity is created and submitted to the purchasing agent for approval of the proposed changes to the original order. The approval must take place before the change order can be sent to the vendor. Because the changes need to be approved, these changes cannot be applied to the original order object. At the same time, once the changes are approved, the application will preserve both the original and new versions of a given business data object, thereby providing an automatic audit trail if needed.

3.3 Context

You are designing a business application that requires that business data objects are constantly available, even when these data objects are in the process of being updated, and the updates can take time in order to be validated and approved. In addition, you need a mechanism to keep track of all changes performed on a given business data object over time.

3.4 Problem

How can existing business data object continue to maintain its original data within a business application while the proposed changes undergo validation, verification and approval. How can we keep track of all changes made to a given business data object over time.

3.5 Forces

- While updates are being performed on the new version of the business data object, the previously approved version must remain available within the given system/application.

- Requested updates which are not approved or which fail validation/verification must not be applied to the existing business data object.
- Business data objects must always be available and accessible within a system, even while the updates on these data objects are being performed.
- A full history of all changes performed on a business data object needs to be preserved and is readily available when needed.

3.6 Solution

The class diagram in Figure 2 illustrates the design for this pattern. When a business data object needs to be modified, a new version of that object is created by making a duplicate copy of the original business data object. The original business data object is available within the system as identified by the *previousVersionObj* reference until changes are completed to the new version and approved, at which point the new version becomes the official business data object within the system, and the previous version is deactivated.

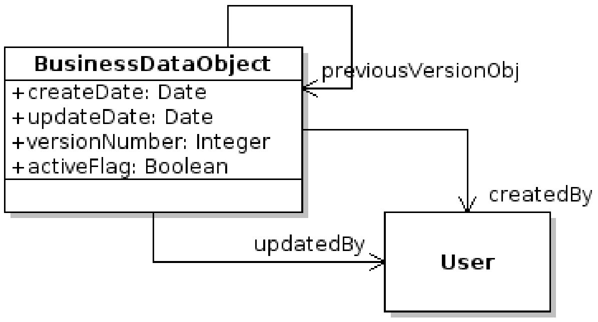

Figure 2 UML class diagram for a Business Data Object with new version attributes

The *activeFlag* represents whether the object is usable or not. The sequence diagram from Figure 3 shows the sequence of operations involved when a change is made to a business object. The original business object creates a new version, which is a duplicate of itself, but with the new changes, then submits the new version object for an approval process that could be an external element of the system. If the approval succeeds, the new object replaces the original one, while keeping a reference to it, which is useful for audit.
The following tables list the procedure using the purchase order example described in the example above.

The original order has a Ship To Code of US001:

Order ID	**Order Version**	**Active Flag**	**Previous Version**	**Create Date**	**Created By**	**Ship To Code**
PO123	1	true		Jan 1, 2015	User123	US001

Table 1 Original Order

We need to change the Ship To code to US023, so a new version of the order is created:

Order ID	**Order Version**	**Active Flag**	**Previous Version**	**Create Date**	**Created By**	**Ship To Code**
PO123	1	true		Jan 1, 2015	User123	US001
PO123	2	false	PO123, Version 1	Jan 2, 2015	User567	US023

Table 2 Change Order In Progress

While the new version is being approved, the original version is still available in the system (its *activeFlag* attribute is set to *true*). Because the original version has already been approved, it can no longer be modified and is therefore a read-only version, thereby preventing any possible branching into more than one new version at a time. The new version is only available for approval, verification and/or validation, but not for general use within the system (*activeFlag* is *false*).

Once the new version is fully approved, it becomes the active read-only data object. The previous version is deactivated, and is no longer available for general use within the system:

Order ID	**Order Version**	**Active Flag**	**Previous Version**	**Create Date**	**Created By**	**Ship To Code**
PO123	1	false		Jan 1, 2015	User123	US001
PO123	2	true	PO123, Version 1	Jan 2, 2015	User567	US023

Table 3 Change Order Completed

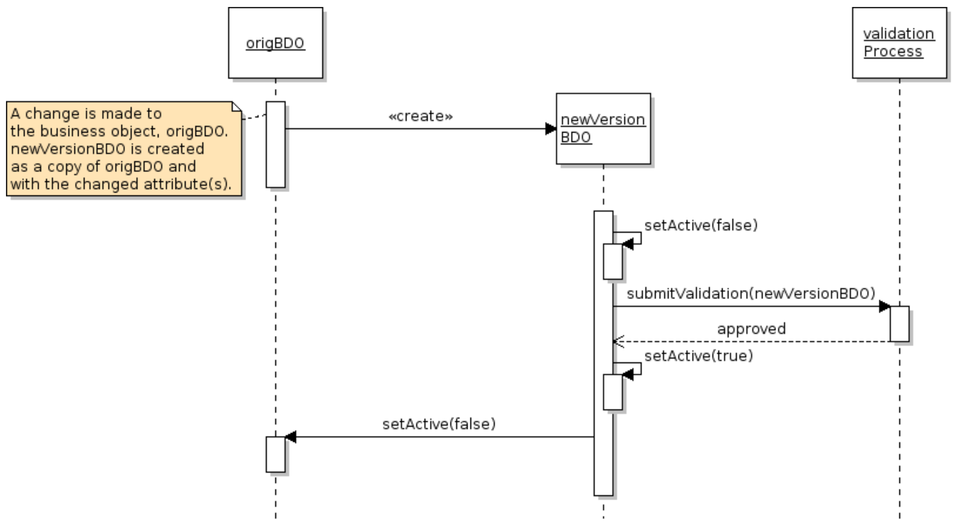
origBDO
validation Process
A change is made to the business object, origBDO. newVersionBDO is created as a copy of origBDO and with the changed attribute(s).
«create»
newVersion BDO
setActive(false)
submitValidation(newVersionBDO)
approved
setActive(true)
setActive(false)

Figure 3 UML sequence diagram for the Business Data Object class when a change is made to the object.

While the object's change approval is pending, the system should lock the object for subsequent changes. Once the changes have been approved, the original data object is replaced with its new version. The original data object is not deleted, but rather set inactive: setActive(fase). This ensures that the original database object is preserved, but at the same time is not available for use as it has been replaced by its newer version.

If on the other hand, the requested changes are denied, then the new version can simply be discarded or archived, and the original version of the data object remains in the system as the most recent version.

3.7 Consequences

All changes are performed on the new version of the business data object, while leaving previous version untouched and in read-only mode. Additionally, once a new version has been initiated, no other new versions are allowed. Only one new version of the business data object can exist at any given time. If the validation/approval process is lengthy, the previous version, or as far as the general audience is concerned, the current version, is still available for usage within a system until new version is verified and/or validated, and subsequently fully approved. If changes to the new version are not approved, it can be deleted or archived for audit, leaving the previous version as the most current one. The most recent fully approved version of the business data object is always available within the system, regardless of what changes are being proposed or are in the process of being applied. All changes performed and approved are part of the new version, thereby creating a full automatic audit trail via previous versions of the given business data object. An easy and convenient link between the most current version and the previous ones is provided via the *previousVersionObj* object reference. Because all of the data is preserved at the version level, the specifics of audit trail requirements need not be determined at design time, thereby making this pattern flexible and dynamic from audit trail requirements perspective. However, due to versioning of the business data object, and the logic required to ensure that the most recent version of the business data object is available to the application and all previous versions are inactive, this pattern might not be an ideal fit for existing applications, where the business data object presentation logic is already in place and might not be easily adaptable to the versioning pattern. Changes to the new version by multiple users are not tracked as part of this pattern. To address the multi-user changes to the new version, it is recommended that this pattern be combined with the Change History Record Pattern, which is covered in the next section.

3.8 Known uses

Many COTS developers actively use a data object versioning in their applications, including SAP and Ariba. Users of these applications are provided with easy access to view and report on all changes performed on a given business data object allowing for seamless audit trail. Ariba, the developer of Procurement and Supply Chain applications, uses the concept of New Version on many of their data objects, including Requisitions, Purchase Orders, Invoices, etc. [8] When a user needs to make a change to a requisition that has already been approved, he or she simply clicks on the Change button, which behind the scenes results in the creation of the new version, which is a duplicate copy of this requisition. Ariba denotes the new requisition number with –V (i.e. V2 for version 2, V3 for version 3, etc.) to inform the end user which version they are working with.

4. THE CHANGE HISTORY RECORD PATTERN

4.1 Intent

The versioning pattern presented in the previous section falls short in a multi-user environments where the changes to a business data object are performed by multiple users, and there is a need to capture the information about who made the changes, when, and what changes were performed, as well as being able to configure at runtime which changes should be recorded and which can be ignored. The Change History Record Pattern offers one such approach.

4.2 Example

An employee gets married, receives a long overdue promotion and a raise. Within an HR system, this change in marital status, promotion and increased salary needs to be updated on the employee record. Because of sensitivity and privacy, the salary should only be updated by someone with a more privileged access than a user who could update marital status and employee level. Thus, at least two users would be updating the same employee record. Each update would need to be recorded and stored for future reference. The recorded data would contain the date and time of the update, the owner of the update, the reason for the update, as well as the old and new values of the updates.

4.3 Context

You are designing a new business application which requires that all changes to existing business data objects are recorded and available for audit, reporting, searches, etc. Often the subject matter experts (SME) such as auditors are the ones who must determine what changes must be recorded, thus the types of changes which will be recorded should also be configurable at run-time, and not hard-coded at design-time.

4.4 Problem

How can a system be designed such that an SME can easily configure a change history recording for any given business data object.

4.5 Forces

- An audit trail is often required for certain business data objects, but the details of which specific changes the audit record must contain are not available at design time.
- Business subject matter experts (SME), and not application designers and developers, should be in control of what data falls under audit trail requirements.

- Business data objects can often be updated multiple times by multiple users during the objects' life cycle, but the number and type of changes that will be performed is not known at design time.
- Records of changes to business data objects must contain detailed information on who performed a given change, when, why, and what change took place (i.e. new value and old value are preserved as part of the record of change).
- Often, an audit trail and historical records must be implemented for business data objects in legacy applications designed without any support for audit trail / historical record keeping.
- If space is a concern, it is beneficial to have a control over which parts of the business data object should be part of audit trail, and which can be ignored.

4.6 Solution

The class diagram in Figure 4 describes the design for this solution. The business object references a set of change record objects. Each change to a business data object – the old value and the new value – and the time of the change are recorded and stored by a *ChangeRecord* object. This record is then linked to the business data object (parent object) via an object reference (*businessObj*) or by some other uniquely identifiable means.

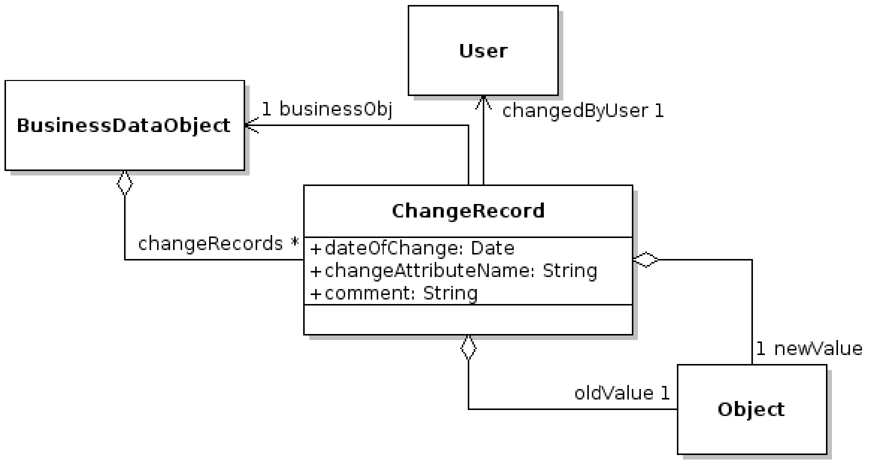

Figure 4 UML class diagram for a business data object with change history records.

Although a history record can be comprised of a wide variety of attributes to address specific business needs, at minimum each change record must have the following attributes and associations:

- Date and time of change. This attribute stores the date/time the change was performed.
- *changedByUser*: reference to the user object initiating the change.
- *changedAttributeName*: the name of the business data object attribute (field) being changed. For example, when employee's salary is raised, this attribute would contain a name of the salary attribute for employee business data object.
- *newValue:*the new value of the business object field that was updated. In the example of employee's salary raise, this attribute would contain the employee's new salary amount. Because the type of the data being updated is not known in advance, the type of New Value attribute should be Object, which in most modern systems can store just about any data type.
- *oldValue*: stores the old value of the field that was updated. In the example of employee's salary raise, this attribute would contain the employee's old salary amount. Because the type of the data being updated is not known in advance, the type of *oldValue* attribute should be Object, too.
- *businessObject*: this reference points to the business data object being updated. There should be no Change Record object without a reference to a valid business data object.
- *comment*: this is an optional String attribute for storing a user entered comment about the reason for the change being performed.

4.7 Consequences

This pattern provides the ability to store any changes performed on the business data object within a given system. It allows for recording of changes to the same business data object by multiple users/processes. It also allows for recording of the identity of the user who performed the change, when, why, and the new and old values of the changed fields. The pattern assumes that the changes performed on the data object are synchronized and only one user/process may perform the changes at any given time. A rollback of the changes can be achieved by combining this pattern with the versioning pattern presented in the previous section, whereby unapproved changes request is denied and the changes are discarded. The pattern does not require that the audit trail requirement be known at design time. Instead, the pattern allows for the audit trail requirements to be highly configurable at run-time. By allowing this high flexibility over the configuration, of the three patterns presented in this paper, this pattern is best suited for business SMEs. The ability to configure which data falls under audit trail, and which does not, the pattern is also sensitive to possible space constrains and the volumes of audit trail and historical records which could bare direct effect on it. The pattern also provides the ability for legacy business data objects which might not have been originally designed with audit trail in mind. Because the pattern allows for high level of configuration, storage requirements and availability should be addressed when large number of changes are present for a given business data object.

4.8 Known Uses

Many COTS developers actively employ change history records in their applications, including SAP and Ariba. All configured changes in these applications are recorded, stored, and accessible, making these applications in compliance with various legal and corporate policies. Ariba applications use a similar approach in their data objects architecture [8]. Each data object that requires historical data tracking is assigned a vector (or a list) type field to store all historical changes performed on the object. Ariba uses the concept of groups, which determine which attributes/fields on a given data object are tracked by historical records when changes are performed on attributes/fields.

5. CONCLUSIONS

The three patterns presented in this paper build on each other's strengths and together provide a design approach for building business data objects that are capable of tracking changes performed on them throughout their life cycle. Our aim was to organize these three patterns in a language neutral and data storage neutral approach. Similar design elements have been used by the enterprise software industry. Continuously changing political, economic and legal conditions dictate that audit trail and change history preservation is included as an integral part of many enterprise applications, and the three patterns presented in this paper make a meaningful contribution to this important, yet often overlooked area of enterprise application design and development.

6. ACKNOWLEDGMENTS

We would like to thank Hans Wegener for his invaluable feedback and support during the shepherding of this paper.

7. REFERENCES

[1] A Guide To The Sarbanes-Oxley Act, The Sarbanes-Oxley Act of 2002, http://www.soxlaw.com/
[2] Thomas C. W., The Rise and Fall of Enron, Journal of Accountancy, April 2002, DOI= http://www.journalofaccountancy.com/Issues/2002/Apr/TheRiseAndFallOfEnron.htm
[3] Fowler M., 2002. Patterns of Enterprise Application Architecture, Addison Wesley
[4] Fowler M., 1997. Analysis Patterns: Reusable Object Models, Addison-Wesley Longman.
[5] Daum B., 2003. Modeling Business Objecs with XML Schema, Morgan Kaufmann Publishers
[6] Eriksson H.-E., Penker M., 2000, Business Modeling with UML: Business Patterns at Work, OMG Press / Wiley Computer Publishing
[7] Adams J., Koushik S., Vasudeva G., Calambos G., 2002, Patterns for e-business: A Strategy for Reuse, IBM Press
[8] Ariba Inc., 2013. DOI=http://www.ariba.com

[9] Wong K. and Bass D., May 23, 2012. SAP to Acquire Ariba for $4.3 Billion in Push Into Cloud. Bloomberg
[10] Wegener H. and Marti R., Slowly Changing Dimensions – a Pattern Language for Coping With Change in Analytical Information Processing, EuroPLoP 2005
[11] Rubis R, Cardei I., Patterns for fine-grain access-controlled business objects, PLoP 2014
[12] Oracle Applications Technical Reference Manuals, Oracle Technology Network, Oracle Corp, http://www.oracle.com/technetwork/apps-tech/index.html

[13] Rubis R. and Cardei I., The Dynamic Business Object Pattern, 20th Conference on Pattern Languages of Programs (PLoP) 2013