ISBN: 9781450308809 (2011). http://doi.acm.org/10.1145/ 2069216.2069255

LIFECYCLE AND GENERATIONAL APPLICATION OF AUTOMATED UPDATES TO MDA EIS APPLICATIONS

Jon E Davis
Curtin Business School
Curtin University
Bentley, 6102, Australia
+61 410 320 956
Jon.Davis@curtin.edu.au

Elizabeth Chang
Curtin Business School
Curtin University
Bentley, 6102, Australia
+61 423 022 745
Elizabeth.Chang@cbs.curtin.edu.au

ABSTRACT
EIS applications are complex and present significant costs and issues during upgrades which can lead user organisations to defer or abandon potential upgrades and cause them to miss out on the business benefits of the upgrade.

Our ongoing development of temporal meta-data EIS applications [1] seeks to avoid or minimise the majority of these upgrade issues by standardising all update procedures to become an updated set or stream of meta-data changes that will be sequentially applied to implement each individual meta-data change in order, for all changes between the previous and current meta-data models.

This update process removes the need from vendors to produce version specific update programs, and fully automates the end user's meta-data EIS application update processes. Collision detection with third party customisations to meta-data EIS application, known as Variant Logic, will be greatly simplified as any potential conflict will be precisely identified in advance, reducing any compatibility effort for the customisations and ensuring timely availability for inclusion with the streamlined meta-data update.

The effort for major EIS updates can be drastically reduced from often months down to days or less with the meta-data update process.

Categories and Subject Descriptors
D.2.13 [**Software Engineering**]: Reusable Software – Domain engineering, Reusable libraries, Reuse models.

General Terms
Algorithms, Management, Documentation, Performance, Design, Reliability, Standardization, Languages, Verification.

Keywords
meta-data, meta-model, variant, logic, EIS, automated update, automated upgrade, version control, version management, software configuration management.

1 INTRODUCTION
By their very nature EIS applications are complex, with wide ranging scope and functionality. They can also present significant costs and issues during upgrades which can lead some user organisations to defer or abandon potential upgrades and cause them to miss out on any of the included business benefits of the upgrade.

There are many contributing factors to the difficulties and costs involved in upgrading traditional EIS applications, such as:

- Customisation of aspects of the EIS application is a common means for user organisations to achieve their preferred functionality. Each customisation has to be seperately reviewed for compatibility with the update and potentially modified.
- Organisations often defer upgrades to reduce costs and application downtime, potentially requiring larger effort during the ultimate upgrade projects.
- Due •Due to the longevity of many EIS applications they may also be internally composed of modules and components using multiple, varied and legacy technologies that have been integrated "under the hood", complicating ongoing integrations and potentially requiring platform installation and migration aspects for each end user.
- EIS applications necessarily cater for broad functionality and will affect a large proportion of the business operations and users requiring a significant level of quality assurance and user education to be successful, as every update is completely unique.

The overall lifecycle costs of maintaining an EIS application are compounded when accounting for all major version upgrades, updates, patches and field fixes that may be released by the application vendor, particularly when the end user has employed customisations that need to be reviewed and may need re-engineering.

Our ongoing development of a temporal meta-data framework for EIS applications seeks to avoid or minimise the majority of these upgrade issues, as an example of the model driven engineering paradigm. A meta-data EIS application is defined and stored as a model, without the need for additional coding, for direct execution by its runtime engine.

A major objective of the framework is to streamline deployment of application updates, which instead of new code, new database objects, and specific and unique migration programs and procedures as typically required, is replaced by an updated set or stream of meta-data that will sequentially execute and implement each individual meta-data change in order, for all changes between the previous and current meta-data model.

With this deployment capability the issue of how many versions or updates need to be progressively applied to a meta-data EIS application is reduced to the one extended update process as all updates can be applied sequentially and as a single process rather than as multiple separate upgrades.

An additional specific objective of the framework is to also provide the capability for end users or third parties to define and create their own application logic, to supplement or replace a vendor's pre-defined application logic, as what we term Variant

Logic [2], to become a variation of the application logic, analogous to customisations in traditionally developed applications.

Variant Logic can be applied to any object defined in a meta-data EIS application; visual objects of the user interface, logical processing objects such as events, functions or workflow, or as data structures.

All integration points between the core meta-data EIS application logic and each Variant Logic instance can be identified as to its independence of any core application changes, and every potential area of logic conflict or collision can be clearly and fully disclosed and documented to the logic definers, to minimise the scope for further review and potential rectification works.

This identification of all changes also extends to the core application meta-data which can provide clear identification of all changes to the end-users to aid in their education of the update impact.

This dual ability to simplify the update process and to clearly identify exactly where logic customisations may be in conflict combine to provide meta-data EIS applications with significantly reduced maintenance effort and costs over the system lifecycle.

2 RELATED WORKS

The following related issues have guided this research to define the deployment and update capabilities and processes of the temporal meta-data framework for EIS applications.

2.1 Software Version Management

Version control is the goal of software configuration management, to ensure the controlled change or development of the software system.

Commencing as a manually managed process, software version management applications have become commonplace to software developers [3] to track the development of the components and manage the baseline of software developments [4] including throughout the various phases of a project [5]. Code changes can be managed to their level of structural organisation within the application as in enterprise level source code management systems such as Microsoft Visual SourceSafe and Team Foundation Server, Surround Software Configuration Management and Collabnet Teamforge as popular examples.

The atomic level required for a meta-data system is the individual object definition within the meta-data EIS application model which needs to be managed at a low level and is also fundamentally tied to direct dynamic execution.

An associated technique for identifying changes between versions of software [6] is a key approach when applied to meta-data and is instrumental to an automated update approach.

2.2 OMG, MDA, MOF and CWM

The aim of the Object Management Group (OMG) is to "provide an open, vendor-neutral approach to the challenge of business and technology change". The OMG represent one of the largest initiatives for Model Driven Engineering (MDE). Their Model Driven Architecture (MDA) initiative is to "separate business and application logic from underlying platform technology" [7].

Their approach is predicated on the design of platform independent models defined primarily with Unified Modelling Language (UML), which can be rendered into a platform specific model with interface definitions to describe how the base model will be implemented on the target platform.

The OMG's Meta Object Facility "provides a metadata management framework, and a set of metadata services to enable the development and interoperability of model and metadata driven systems". Its intention is to promote cross platform access to independent modelling systems and definitions in a common format as an agent of sharing and reuse.

The OMG's Common Warehouse Metamodel is an associated technology to support the common storage of UML and MOF models to be accessed by modelling and coding toolsets.

The OMG supports industry developers of supporting toolsets, as well as user developers of the technologies.

The goal of the OMG is interoperability, and the tools and technologies are primarily aimed at highly technical analysts and developers. Our objective for the meta-data EIS application includes technical analysts for the vendors or logic definers but is primarily targeted at business user and operational optimisation.

2.3 Software Update and Deployment

Software updates for applications have traditionally been released in a form of hard media that is distributed to the end user although this has largely been superseded by electronic distribution via the internet.

For smaller consumer and utility software systems the update often consists of a specific update program and instructions, or alternatively a replacement program that uninstalls the previous version and installs the latest version. Both will operate largely automatically with minimal user input required.

Larger EIS/ERP style systems tend to utilise either the version update process or install the new version and attempt to migrate the data and configuration from the previous version installation.

The larger and more complex a system is the less likely that automated updates will complete successfully as less effort and quality assurance seems to be expended on producing each specific update program than on the primary software product [8], exacerbating existing common issues with system development quality assurance [9].

Managers of EIS upgrades attest to the often extensive projects required for particularly major version EIS upgrades which can require months of effort and considerable expense.

The minimisation of effort for updating meta-data EIS applications is a major objective of our research.

2.4 Application Customisation and Rework

In the best of situations some major EIS upgrades may be performed relatively quickly although one of the pre-requisites for this success must be a virtually out-of-the-box implementation without any customisations.

It has become commonplace for end user organisations to engage the vendor or authorised third parties to develop specific customisations for their user requirements to become embedded within a new localised version of the application. Notwithstanding the initial expense, additional review and potential re-engineering is required for each customisation when

the EIS is upgraded to ensure ongoing compatibility, which adds often considerable time and expense to each upgrade. [10]

Customisation of EIS systems for the local environment has become a fact of life for many end user organisations, and reducing the impact of the use of customisations through the maintenance lifecycle is another major objective of our research.

2.5 Model Driven Engineering

Alternatives to the common process of hard coded application logic are provided by ongoing Model Driven Engineering (MDE) which is a generic term for software development that involves the creation of an abstract model and how it is transformed to a working implementation [11].

Utilising a meta-data model based interpretation of the application specification allows applications to be executed using any simultaneous combination of platforms that are supported by the components of the runtime engine, providing a progression towards complete platform independence.

A significant proportion of the works to date have involved modelling which contributes more directly to streamlining code generation, processes that are directly aimed for and dependent on highly technical programmers. [12] base their works on the UML 2 specification to seek to reduce coding and transform models of business processes into executable forms.

The visual structure meta-data is used to construct the appearance of the application as presented by the user interface runtime components to the users. The program flow meta-data is used to define the user interface and local platform actions and procedures that are executed in response to user actions and other data changes. The data dictionary meta-data is used to define the requirements of the database schema and the data changes required in response to user actions and other data changes.

Such a model is the goal of our temporal meta-model framework for EIS applications [13].

Every aspect of the EIS application functionality is a component of the meta-data model, whether it is identified as core application meta-data produced by the original vendor, or whether it is a modification or extension produced by a user or third party as Variant Logic. Meta-data version updates can always be clearly identified by a comparison of the meta-data between two time states and then re-producing the sequence of meta-data changes to apply to the meta-data model to be updated.

3 AUTOMATIC APPLICATION UPDATE WITH USER CUSTOMISATIONS

Our ongoing development of a temporal meta-data framework for EIS applications seeks to remove the need for hard coding by technical developers and transform the responsibility of defining application logic to business analysts, knowledge engineers or even business end users.

Similarly, the application update process can be greatly simplified as we remove the need for specific version upgrade programs and procedures for every minor or major upgrade, patch or field fix. Updates are always a series of identified changed meta-data that is applied sequentially to the target meta-data application until all changes have been applied.

In a similar way that the meta-data model is always executed by users using the same runtime engine, every meta-data update is processed by a simple update engine that updates the meta-data model and facilitates any associated database operations where data or data definitions may need to be modified. Any data locking and data migration requirements are managed automatically by the update engine which can also allow the updates to be enacted on live systems if required.

With this deployment capability the issue of how many versions or updates need to be progressively applied to a meta-data EIS application is reduced to the one extended update process as all updates can be applied sequentially and as a single process rather than as multiple separate upgrades.

A unique feature of the application of temporal data management techniques to the atomic meta-data elements of the meta-data EIS application can also provide for a complete temporal execution of meta-data EIS applications by maintaining a perfect synchronisation of historical data with the historical application states. The temporal meta-data framework can allow meta-data EIS applications to execute across time, regardless of the meta-data EIS application version changes that have occurred. This feature also supplements any manual testing of an updated meta-data EIS application as it provides comparative easy access to the pre-updated version of the meta-data EIS application.

User or third party customisation of the meta-data EIS application is provided as Variant Logic, to become a variation of the application logic, and can be applied to any object defined in a meta-data EIS application whether; visual objects of the user interface, logical processing objects such as events, functions or workflow, or as data structures.

As the Variant Logic itself consists of meta-data as part of the extended model, then all integration points between the core meta-data EIS application logic and each Variant Logic instance can be fully determined and identified as to whether the Variant Logic remains independent of any core application changes, or may have some associated logic areas required for review, or where the application has been updated to cause a conflict for an existing customisation.

Where the Variant Logic is not fully identified as compatible with the updated application meta-data logic, a key benefit is that every potential area of logic conflict or collision can be clearly and fully disclosed and documented to the logic definers, to minimise the scope for further review and potential rectification.

This identification of all changes also extends to the core application meta-data which can provide clear identification of all changes to the end-users.

3.1 Meta-Data Version Control Framework

In our meta-data EIS application model, version control needs to be applied to only two of the aspects of the model; the overall Application Model object, and to any Logic Variant object.

The temporal meta-data management aspects of the model internally tracks all changes that are made to any of the model's meta-data whether as core application changes, user or third party customisations or Variant Logic to identify the constituent meta-data for each defined version.

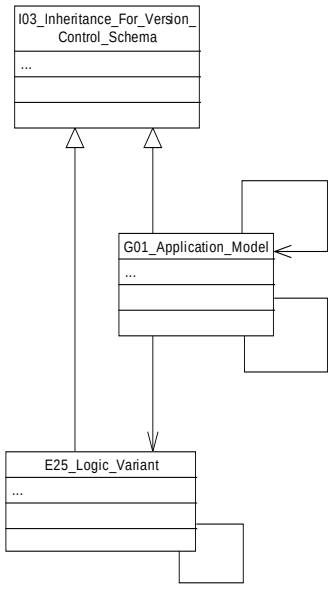

Figure 1: Applicability of Version Control in meta-data EIS application model.

Version Control (see Figure 1) uses the following classes to model the definition of the change access:

- **Inheritance For Version Control Schema**: is inherited to objects requiring specific version control attributes.
- **Application Model**: is the high level identifier of the application as modelled in the meta-data EIS application. This identifies and groups all of the application's meta-data objects.
- **Logic Variant**: is a designated identifier to group all of the logic changes together into a practical set as an instance of Variant Logic. The best use of a Logic Variant would be to group the associated changes of a set of new functionality for a specific purpose.

The Version Control classes facilitate the identification and classification of the meta-data into the logical groupings that we humans understand as specific versions. Internally, it is the ongoing temporal management of the meta-data that maintains the true atomic history of the application evolution by tracking each individual logic change in the meta-data model.

The Application Model object, representing the overall grouping object for the model meta-data, can be divided into any hierarchy of sub-Applications to classify and organise the core application meta-data into modules and sub-modules as required (see figure 2).

The sub-Application grouping is to facilitate the logical grouping of functionality by vendors or logic definers of the meta-data EIS application models. Sub-Applications provide a suitable breakdown for the deployment and tracking of individual modules and as an additional selection criteria for assigning security access but have no other logical limitations within the meta-data model.

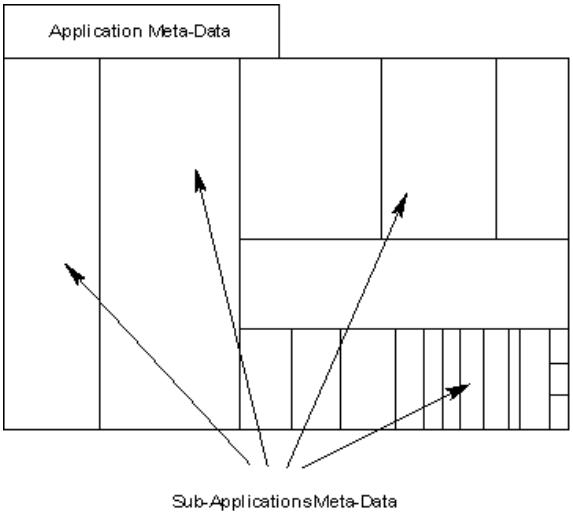

Figure 2: Core Application meta-Data composed of Sub-Application meta-data.

Whilst the core application as delivered by the vendor or logic definer may initially represent the totality of the meta-data defining all logical functionality, the meta-data EIS application framework also permits additional meta-data to be defined for new local functionality.

Although any additionally defined meta-data can be defined by any other authorised user or third party, all additional meta-data must also be associated with the Application Model object and be subject to local authorisation and access of the core application environment.

The logic definer authorisation processes are governed by the following principles:

- All original meta-data is owned by the identified core logic definer, usually at the highest authorisation level.
- Additional logic definers can be defined with lower level authorisations.
- Meta-data objects owned by one logic definer cannot be modified by a different logic definer, to ensure application semantic integrity.
- Any logic definer can define new meta-data, reference and invoke meta-data owned by other logic definers, and modify undefined meta-data attributes of meta-data owned by other logic definers where this functionality has not been restricted.
- Meta-data defined by a higher level logic definer always over-rides any other identical meta-data definition created by a lower-level logic definer – this aspect will be further discussed during update collision detection.

There is no limitation on what logical functionality can be defined by users or third parties other than any authorisation limitations that may be imposed on access to existing objects. Minor additions or entire add-on modules or applications can be defined to supplement a meta-data EIS application.

The final aspect of user or third party customisation is provided as Variant Logic, which is a modified copy of an aspect of the core application logic that becomes an alternative variation of

the application logic. It too can be applied to any object defined in a meta-data EIS application.

There can be multiple and different Variant Logic sets involving the same meta-data as different users may choose and be authorised, in both the security and semantic domains, to prefer separate alternate optimised logic for their specific usage under their local conditions.

The scope of Variant Logic is also unlimited, subject to ongoing access authorisations, other than any logic that is restricted by the original meta-data logic definer. Restrictions are typically imposed to maintain information processing standards for key meta-data definitions.

While Variant Logic is defined to alter existing application functionality, it is also defined on existing application meta-data objects in order to define access to user and third party customisations e.g. adding navigation menu items, or adding buttons to user interface screens, to invoke new functionality.

Figure 3 illustrates the extended meta-data model that includes the core application meta-data, user and third party customisations meta-data, and the Variant Logic meta-data extensions.

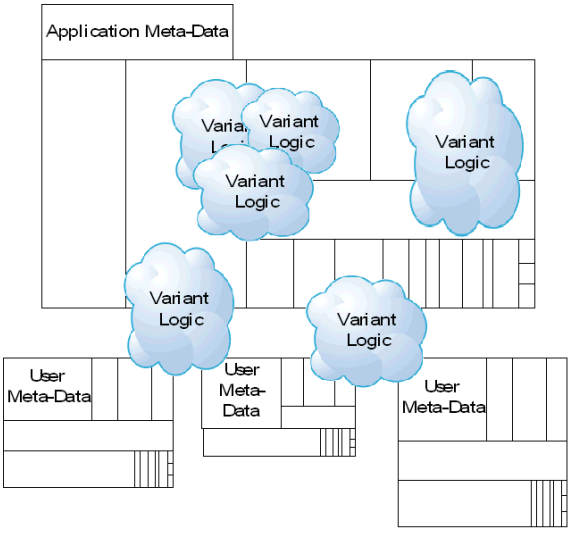

Figure 3: Additional custom user meta-data and Variant Logic.

In summary, Version Control for meta-data EIS applications operates under the following principles:

- Core application meta-data is provided as an Application Model and managed as sub-Applications, tracking the version of each sub-Application and its defined meta-data.
- Additional new application logic can be defined as meta-data by users and third parties and managed as sub-Applications.
- Any updates to application logic meta-data are tracked as belonging to an updated sub-Application with its associated version information.
- Meta-data changes are managed as sequential builds and may include changes from any combination of sub-Applications.
- Alternate parallel versions of existing application meta-data can be defined by users and third parties to provide modified functionality and to reference new application meta-data.

In traditional application development the updates are provided as replacement executable files, database migrations and upgrade programs which provide the outcomes of the changes but rarely identify all changes to the users except through perhaps a prepared text summary. Even the application vendor's internal programming staff may not fully identify all of the programming changes unless they utilise comprehensive internal version control management that integrates across all of the implemented technologies.

The meta-data EIS application can clearly identify all changes, the order that they were made, and the impact and object relationship of the changes. In the following sections, we will see how updating the meta-data EIS application is performed using greatly simplified and standard processes that remove the need for complex individual upgrade procedures required for traditional developments.

3.2 Defining the Meta-Data Update

There are two aspects of defining the scope of the meta-data changes that are to be applied as part of the update process:

- **Continuity**: ensure that meta-data changes apply to the end user organisation's current version,
- **Content**: select all meta-data changes that are appropriate for the selected meta-data update.

Without appropriate validity applied to these aspects the meta-data update may fail and leave the meta-data EIS application environment with an unstable model, although the update validation procedures would detect and either reject or rollback from incomplete or erroneous meta-data update sequences (see section 3.3).

Continuity is ensured by the meta-data definer sequentially identifying the build release of all versions of its application meta-data independent of the scope of the meta-data changes of that release. As meta-data updates, which may include changes to both the application logic and to the underlying data structures of the modelled application, must be applied continuously this build identification against each change in the meta-data update sequence guarantees continuity is maintained.

The build identification also allows for greater flexibility in the availability and application of the meta-data updates by releasing multi-version meta-data updates that can be applied by the end user in different ways (see Figure 4);

- **Update Start**: for an end user currently at build N of a meta-data EIS application, a multi-version release can include any previous build meta-data which will be ignored by the meta-data updater which would only commence the update with the meta-data update items from build N+1 in the multi-version update stream,
- **Update End**: an end user can choose to cease or hold the meta-data update at any available build level greater than their current build level. This may be desirable depending on internal update and test policies, or potentially due to available downtime windows if some builds involved extensive functional changes or intensive data changes.

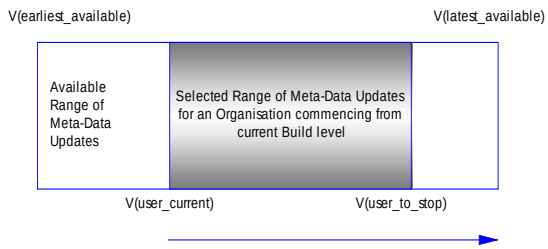

Figure 4: Optional range of selected meta-data update.

The content of the changed meta-data for each new build level is based on the meta-data changes as defined in a vendor's or other logic definer's defined internal development systems.

Similarly to traditional development, a meta-data application logic definer must also maintain its application development, aka meta-data definition processes, according to efficient internal version control procedures for software engineering. This may involve any distributed or centralised combination of logic definer and test servers where the scope of the meta-data logic changes have been segmented, distributed, combined and otherwise managed to its final approved state.

Each approved meta-data change to an existing meta-data model will become part of an identified build set of meta-data changes.

The scope of any meta-data build set may include meta-data from multiple sub-Applications or be specific to a single functional area – this is at the discretion of the logic definer.

Also for commercial reasons, a vendor may wish to place additional restrictions on the included scope of any build set release that is provided as an update to its customers. E.g. to include only the meta-data for particular sub-Applications that are licensed to some customers. The only caveat is that where a logic definer chooses to limit the scope of the build release that they ensure the logical consistency of the released build set to ensure compatibility with the stated release target users (see Figure 5).

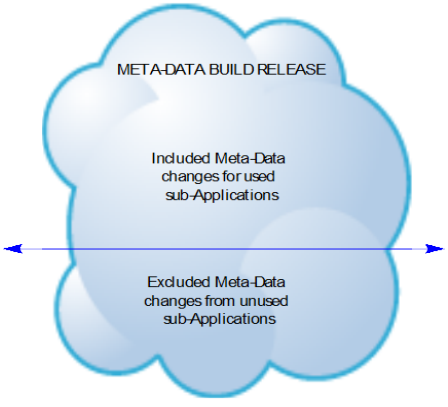

Figure 5: Optional scope restricted build for a meta-data update.

A consequence may be that a particular released build set may be a null set and include no specific updates, as a valid release. This build set must still be included as part of the overall sequential lifecycle updates to ensure overall continuity is maintained.

3.3 Automated Meta-Data Update and User Customisation Detection

Traditional applications require the source code to be compiled and packaged into the set of executable application files, which then need to be made available to the users for testing and operational access. The required combination of application testing, distribution, organisation testing, customisation re-engineering and deployment all contribute to delays in the effective release of the application software. These delays will always be exacerbated for the larger and more complex EIS software due to the organisational criticality of the EIS and its need for extensive testing, hence the current reality of real world EIS implementations that typically require several months to implement new or upgraded major versions.

Additional complication occurs when a user organisation has also implemented their own customisations to the EIS, a common occurrence which can often require major rework of the customisations to ensure operation of or compatibility with the updated EIS. It is rarely an inexpensive task which often results in organisations deliberately skipping on many minor and even some major releases in order to reduce costs – at the additional business cost of missing out on any of the positive benefits that may be provided by the update.

As discussed in the previous section, the source update to the meta-data EIS application is an ordered sequence of meta-data changes classified by the logic definer's build release. The meta-data EIS application can drastically reduce these delays due to the wholesale change in the development methodology lifecycle and the unique meta-data update deployment model, which can reduce the overall deployment delays down to at most days or even virtually instantaneous distribution and update.

It also becomes possible to execute updates on a live system, at the risk of some performance degradation and periodic functional locking, although prudence would always suggest first deploying the updates to a test meta-data EIS application environment first. While this is always a practical environment to maintain, the meta-data EIS application lifecycle and update processes provide great optimisations and significant savings in time and resources.

An authorised meta-data update may also over-ride other identical meta-data functionality defined by other lower-level logic definers. The meta-data update process can identify these occurrences during the update and prepare a report of potential changes to lower-level meta-data so that their meta-data definers can review and modify their meta-data to ensure continued semantic integrity. Note that this update report becomes a very specific report on how any higher-level meta-data update has impacted on other third party pre-defined lower-level meta-data, and can clearly avoid the major re-engineering works on customisations that occur in the traditional EIS environment.

Similarly, as the updated meta-data is clearly identified, auto generated descriptions of the affected areas of the meta-data application, as represented by the changed meta-data, can be readily provided. Additionally, auto-generated online and offline help files and user documentation can be created to assist users with the exact nature of the transition.

As an aid to forensic analysis of an organisation's EIS data and contributing transactions, the meta-data EIS application in conjunction with the features of temporal meta-data management can also provide an unlimited facility in replaying

and reviewing the nature and effects of any transactions that have occurred in the meta-data EIS application.

All transaction executions are recorded as part of the audit tracking provided by temporal data management and the subsequent results of changes to the data base are recorded by the temporal data management features, plus any changes to the meta-data EIS application are tracked by the similar temporal meta-data management features. At any time, the authorised forensic analyst can effectively review and replay the previous transaction, called a Temporal Rollback, or review and replay the next transaction, called a Temporal Rollforward.

Each request for a Temporal Rollback or Temporal Rollforward effectively selects and changes the current view in the temporal application window for that user to the requested temporal view as had been executed as a result of the requested transaction, either before or after the transaction.

The ability to execute such Temporal Rollback or Temporal Rollforward operations throughout the entire temporal application window of the meta-data EIS is a unique feature of the temporal meta-data framework. These operations are seamlessly provided without any of the temporal limitations that are typically imposed by non-temporal applications, which further exacerbate the practical access limitations due to disparate or non-existent previous historical version implementations of traditional EIS applications.

In order to perform the meta-data update, the update engine processes the meta-data update stream with the following process:

- The end user managing the update specifies the end build reference for this update process if the meta-data update is a multi-version update, and specifies if live user sessions are to be permitted during the update process. Any update can initially be run in simulation mode which simply identifies all proposed changes but implements none – these changes can be used as the basis of planning the update, preparing users for functionality changes, and allowing logic definers to preview potential conflicts with any Logic Variants they have created.
- If the starting build reference of the meta-data update stream is greater than the current meta-data EIS applications build reference + 1 then the update is abandoned, as the update cannot provide continuity, otherwise
- Progress through the meta-data update stream in sequence until the first meta-data change where the build reference is equal to the current meta-data EIS applications build reference + 1,
- While the update build reference is less than or equal to the selected end build reference for this update process each sequential meta-data change.
- If errors such as data stream checksums, or build references are skipped in the update, or references to non-existent objects occurs, then the update needs to be aborted and rolled back to either the initial state or the last completed build reference as the end user selects.
- Prior to each individual build reference update, a map of all affected meta-data objects is pre-scanned and created so that appropriate locking can be sought from existing user sessions and invoked for future user sessions if live access is permitted during the update.
- The following update process occurs for each meta-data change:
    - If the update is of a visual or logical object type, the change is applied directly to the meta-data object definitions.
    - Otherwise if the update is of a data definition object type then the change is applied and any associated flow through effects on the underlying data structures. It is possible that some data definition changes may cause a temporary error status to exist due to then unresolved compatibility links between usage instances of the change meta-data object. It is expected that subsequent meta-data updates within the same build would resolve all of these interim errors as each compatibility issue was then resolved, exactly as the meta-data editor would have informed the logic definer whilst making the original meta-data changes.
    - Each update checks if the scope of the change conflicts with any existing Logic Variant that has been defined by any other logic definers. A conflict does not prevent the update but the conflict is noted for communication to the logic definer for review of the effect on their Logic Variant.
- Upon completion of all updates for a build, any unresolved compatibility links between meta-data objects will be notified if they have occurred and the end user can choose to rollback the changes for that build.
- Upon completion of all updates the meta-data EIS application can be made available for immediate use, or typically for a series of end user testing and allowing logic definers to provide any required meta-data changes to Logic Variants that may have been affected by the update.

The meta-data EIS application provides a drastic simplification of the update process for both the vendors and end user organisations. Many meta-data changes will have minimal effect on a live system, although the changed functional areas will be locked automatically while the build update occurs.

The simplification of managing user customisations by identifying only the potentially affected logical components will also reduce any effort required to ensure compatibility. The ability to identify these conflicts before any update is performed can ensure that any rectifying meta-data changes to the Logic Variants are prepared in advance to be applied immediately following the automated update process.

4 CONCLUSIONS

The temporal meta-data framework for EIS applications offers many unique benefits that can significantly reduce the effort for both vendors and end users of meta-data EIS applications in providing and applying version updates.

While our separate analyses have shown that meta-data EIS applications can have proportionally significantly lower lifecycle costs compared to traditionally developed EIS applications (circa 15%), we believe that the automated update capability alone can provide substantial additional tangible efficiency savings, particularly in a highly customised environment, due to:

- The internal mapping between meta-data objects in a meta-data EIS application identifies all relationships and uses of the meta-data objects which aids in identifying impact analysis and tracking syntactic compatibility during logic definition, reducing the instance of basic logical errors being deployed.
- Vendors no longer need to produce dedicated version specific update programs and procedures as the meta-data changes are automatically applied, reducing their cost of meta-data application development, and minimising the scope of induced migration errors – a common update engine is always used.
- End user organisations have more direct knowledge of the changed functionality due to the update simulation which identifies every change. This allows more informed planning of end user resources for clearly focussed testing and training.
- End user organisations can choose how many builds to update and merge updates to reduce overall update overhead.
- Logic definers can be provided with the precise definition of any conflicts between their Logic Variants and the updated meta-data EIS application, reducing the effort in updating the customisations and given advance notice to ensure the timely availability of updated Logic Variants to complete the overall EIS update.
- End user organisations can optionally choose to allow live access to the meta-data EIS application during the updates, reducing overall inavailability and functional group downtime losses.
- Substantial reductions in the overall upgrade project efforts.

The meta-data EIS application has significant potential in greatly reduced lifecycle definition costs and functional advantage due to features such as Variant Logic providing simplified local customisation. Further significant lifecycle efficiencies and cost reductions are available for both vendors and end user organisations with the automated meta-data update processes, minimising update effort, time and costs and maximising end user uptime and the availability of new business functionality.

5 REFERENCES

[1] Davis, J., Chang, E., 2011. Temporal meta-data management for model driven applications, In Proceedings of 13*th International Conference on Enterprise Information Systems*, Beijing, China, June 2011.

[2] Davis, J., Chang, E., 2011. Variant logic meta-data management for model driven applications, In Proceedings of 13*th International Conference on Enterprise Information Systems*, Beijing, China, June 2011.

[3] De Alwis, B., Sillito, J., 2009. Why are software projects moving from centralized to decentralized version control systems ? In *CHASE '09 Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*. 2009.

[4] Ren, Y., Xing, T., Quan, Q., Zhao, Y., 2010. Software Configuration Management of Version Control Study Based on Baseline. In *Proceedings of 3rd International Conference on Information Management, Innovation Management and Industrial Engineering*. Nov 2010. Vol 4. Pp118-.

[5] Kaur, P., Singh, H., 2009. Version Management and Composition of Software Components in Different Phases of the Software Development Life Cycle. In *ACM Sigsoft Software Engineering Notes*, Jul 2009. Vol 34. Iss 4. Pp493-.

[6] Steinholtz, B., Walden, K., 1987. Automatic Identification of Software System Differences. In *IEEE Transactions on Software Engineering*, Apr 1987. Vol SE-13. Iss 4. Pp493-.

[7] OMG, 2010. OMG Model Driven Architecture. In *http://www.omg.org/mda/*, 2010.

[8] Jansen, S., Brinkkemper, S., Helms, R., 2008. Benchmarking the Customer Configuration Updating Practices of Product Software Vendors. In *Proceedings of the 7th International Conference on Compostion Based Software Systems*, Feb 2008. Pp82.

[9] Brown, A., 2004. Oops! Coping With Human Error in IT. In *Queue – System Failures*, Nov 2004. Vol 2. Iss 8.

[10] Dittrich, Y., Vaucouleur, S., Giff, S., 2009. ERP Customisation as Software Engineering: Knowledge Sharing and Cooperation. In *IEEE Software*, Nov/Dec 2009, Vol 26, Iss 6, pp41-.

[11] Schmidt, D., 2006. Introduction Model-Driven Engineering. In *IEEE Computer Science*, Feb 2006, Vol 39, No.2, pp25-31.

[12] Fabra, J., Pena, J., Ruiz-Cortez, A., Ezpeleta, J., 2008. Enabling the Evolution of Service-Oriented Solutions Using an UML2 Profile and a Reference Petri Nets Execution Platform. In *Proceedings of the 3rd International Conference on Internet and Web Applications and Services*, June 2008, pp198-.

[13] Davis, J., Tierney, A., Chang, E., 2004. Meta-data framework for EIS specification, In *6th International Conference on Enterprise Information Systems*, Porto, Portugal, April 2004.