Automatic Application Update with User Customisation Integration and Collision Detection for Model Driven Applications

Jon Davis and Elizabeth Chang

***Abstract*—Enterprise Information Systems (EIS) applications are complex and incur significant costs and effort during version upgrades, which are further exacerbated by any customizations.**

Our temporal meta-data framework for EIS applications avoids or minimizes these upgrade issues by standardizing all update procedures as an updated set or stream of individual meta-data changes. Each change is applied sequentially for all changes between the previous and current meta-data models. Meta-data changes include the core application functionality plus meta-data changes for any local or unique customizations.

The automated update process removes the need from developers to produce version specific update programs, and simplifies the end user's meta-data EIS application update processes. Collision detection with third party customizations, known as Variant Logic, can precisely identify any potential conflict in advance, avoiding or reducing compatibility rework effort for customizations.

***Index Terms*—meta-model, automated update, variant, logic, EIS**

I. Introduction

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

- Each customization has to be separately reviewed for compatibility with the update and potentially modified.
- Organizations often defer upgrades to reduce costs and application downtime.
- Due to the longevity of many EIS applications they may also be internally composed of multiple legacy technologies that have been integrated potentially requiring platform installation and migration aspects.
- EIS applications will affect a large proportion of the business operations requiring a significant level of quality assurance and user education to be successful.

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 customizations that need to be reviewed and may need re-engineering.

A major objective of the temporal meta-data framework [1] 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 a stream of discrete meta-data changes.

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 customizations 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 minimize the scope for further review and potential rectification works.

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

II. Related Works

The following related issues have guided this research to define the deployment and update capabilities and processes

Manuscript received July 12, 2011.

Jon Davis is with Digital Ecosystems and Business Intelligence Institute (DEBII), Curtin University, Enterprise Unit 4, Technology Park, Bentley 6102, Australia (phone: +61 410 320 956; e-mail: jed1@coxdavis.name).

Elizabeth Chang is also with Digital Ecosystems and Business Intelligence Institute (DEBII), Curtin University (phone: +61 8 9266 7085; fax: +61 8 9266 4846; e-mail: Elizabeth.Chang@cbs.curtin.edu.au).

ISBN: 978-988-19251-7-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

of the temporal meta-data framework for EIS applications.

A. Software Version Management

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

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.

B. 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].

The OMG's Meta Object Facility (MOF) "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 modeling systems and definitions in a common format as an agent of sharing and reuse.

The OMG's Common Warehouse Metamodel (CWM) is an associated technology to support the common storage of Unified Modeling Language (UML) and MOF models to be accessed by modeling and coding toolsets.

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 optimization.

C. Software Update and Deployment

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

D. Application Customization and Rework

It has become commonplace for end user organizations to engage the vendor or authorized third parties to develop specific customizations for their user requirements to become embedded within a new localized version of the application. Notwithstanding the initial expense, additional review and potential re-engineering is required for each customization when the EIS is upgraded to ensure ongoing compatibility, which adds often considerable time and expense to each upgrade. [10]

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

E. 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].

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.

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.

III. 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.

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

ISBN: 978-988-19251-7-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

update process as all updates can be applied sequentially and as a single process rather than as multiple separate upgrades.

This also provides clear identification of all application changes that will be made to the end-users.

A. 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 customizations or Variant Logic to identify the constituent meta-data for each defined version.

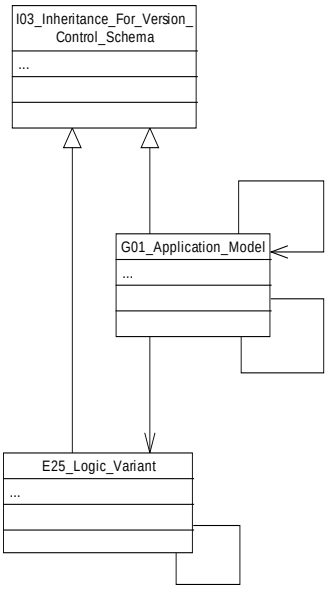

Fig. 1. Applicability of Version Control in meta-data EIS application model.

The primary Version Control classes (see Figure 1) 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 organize 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.

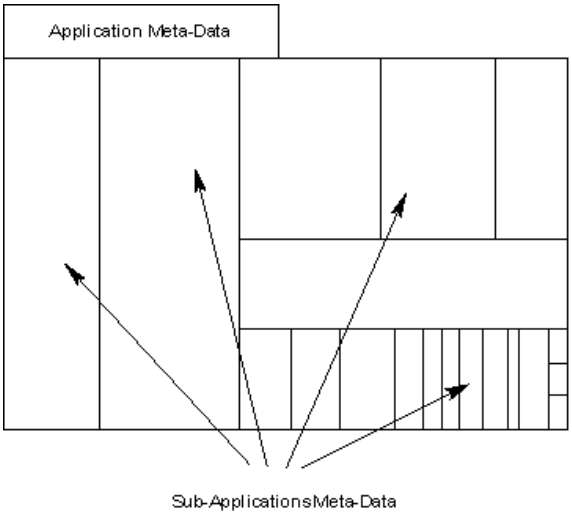

Fig. 2. Core Application meta-Data composed of Sub-Application meta-data.

Any additionally defined meta-data can be defined by any other authorized user or third party. All additional meta-data must also be associated with the Application Model object and be subject to local authorization and access of the core application environment.

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

- All original meta-data is owned by the identified core logic definer, usually at the highest authorization level.
- Additional logic definers can be defined with lower level authorizations.
- 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 authorization 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 customization 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 authorized, in both the security and semantic domains, to prefer separate alternate optimized logic for their specific usage under their local conditions.

ISBN: 978-988-19251-7-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

The scope of Variant Logic is also unlimited, subject to ongoing access authorizations, 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 customizations 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 customizations meta-data, and the Variant Logic meta-data extensions.

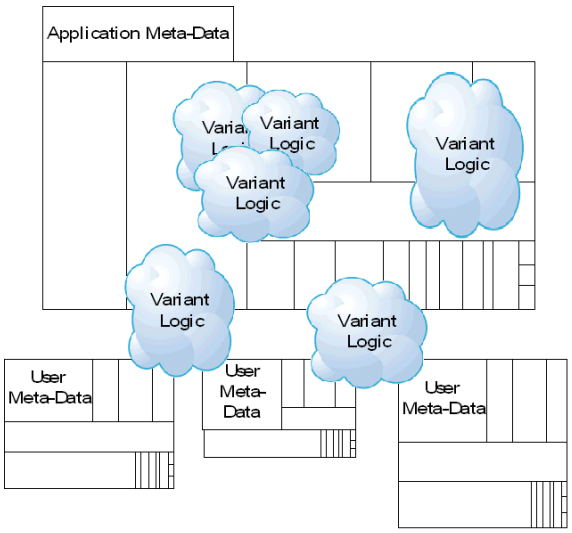

Fig. 3. Additional custom user meta-data and Variant Logic.

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 utilize 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.

B. *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 organization's current version,
- *Content*: select all meta-data changes that are appropriate for the selected meta-data update.

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 modeled 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. 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.

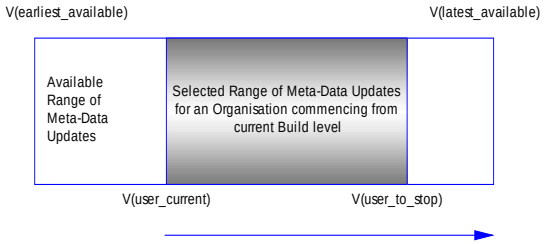

Fig. 4. Optional range of selected meta-data update.

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 centralized 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.

ISBN: 978-988-19251-7-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

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).

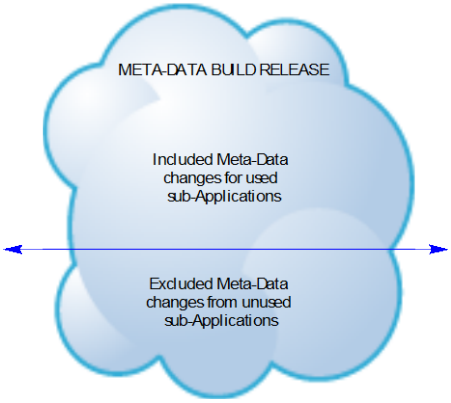

Fig. 5. Optional scope restricted build for a meta-data update.

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.

C. Automated Meta-Data Update and User Customization Detection

Complication occurs when a user organization has also implemented their own customizations to the EIS, a common occurrence which can often require major rework of the customizations to ensure operation of or compatibility with the updated EIS.

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 the overall deployment delays down to at most days or even a 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.

An authorized 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.

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.

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

- The end build reference for this update process is specified if the meta-data update is a multi-version update, also whether live user sessions are to be permitted during the update process. Any update can initially be run in simulation mode to identify all proposed changes to aid update planning preview potential conflicts with any Logic Variants.
- Prior to each individual build reference update, a simulation of all affected meta-data objects is pre-scanned to facilitate object locking from existing user sessions if live access is permitted during the update.
- Out of sequence build references are not permitted, as the update cannot provide continuity, otherwise
- Progress through the meta-data update stream in sequence until the first meta-data change of the correct build reference,
- Process each sequential meta-data change of each updated build reference.
- Errors can be aborted and invoke rollback to either the initial state or the last completed build reference.
- 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.
  - Each update checks if the scope of the change conflicts with any existing Logic Variant that has been defined by any other logic definers for communication to the logic definer.
- 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 organizations.

IV. CONCLUSIONS

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 customized environment, due to:

- The internal mapping between meta-data objects in a

ISBN: 978-988-19251-7-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

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 minimizing the scope of induced migration errors – a common update engine is always used.
- End user organizations 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 focused testing and training.
- End user organizations 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 customizations and given advance notice to ensure the timely availability of updated Logic Variants to complete the overall EIS update.
- End user organizations can optionally choose to allow live access to the meta-data EIS application during the updates, reducing overall unavailability and functional group downtime losses.
- Substantial reductions in the overall upgrade project efforts.

REFERENCES

[1] J. Davis, E. Chang, "Temporal meta-data management for model driven applications", in *Proceedings of 13th International Conference on Enterprise Information Systems*, Beijing, 2011, pp. 376-379.

[2] J. Davis, E. Chang, "Variant logic meta-data management for model driven applications", in *Proceedings of 13th International Conference on Enterprise Information Systems*, Beijing, 2011, pp. 395-400.

[3] B. De Alwis, J. Sillito, "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*, Vancouver, 2009, pp. 36-39.

[4] Y. Ren, T. Xing, Q. Quan, Y. Zhao, "Software configuration management of version control study based on baseline", in *Proceedings of 3rd International Conference on Information Management, Innovation Management and Industrial Engineering*, Kunming, 2010, vol. 4, pp. 118-121.

[5] P. Kaur, H. Singh, "Version management and composition of software components in different phases of the software development life cycle", in *ACM Sigsoft Software Engineering Notes*, 2009, vol 34, iss. 4, pp. 493-501.

[6] B. Steinholtz, K. Walden, "Automatic identification of software system differences", in *IEEE Transactions on Software Engineering*, 1987, vol. SE-13, iss. 4, pp. 493-497.

[7] Object management Group. (2010). OMG Model Driven Architecture. Available: http://www.omg.org/mda

[8] S. Jansen, S. Brinkkemper, R. Helms, "Benchmarking the customer configuration updating practices of product software vendors", in *ICCBSS '08 Proceedings of the 7th International Conference on Composition Based Software Systems*, Madrid, 2008, pp. 82-91.

[9] A. Brown, "Oops! Coping with human error in IT", in *ACM Queue – System Failures*, 2004, vol. 2, iss. 8.

[10] Y. Dittrich, S. Vaucouleur, S. Giff, "ERP customisation as software engineering: knowledge sharing and cooperation", in *IEEE Software*, 2009, vol. 26, iss. 6, pp. 41-47.

[11] D. Schmidt, "Introduction model-driven Engineering", in *IEEE Computer*, 2006, vol. 39, iss. 2, pp. 25-31.

[12] J. Fabra, J. Pena, A. Ruiz-Cortez, J. Ezpeleta, "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*, Athens, 2008, pp. 198-204.

[13] J. Davis, A. Tierney, E. Chang, "Meta-data framework for EIS specification", in *Proceedings of 6th International Conference on Enterprise Information Systems*, Porto, 2004, pp. 451-456.

ISBN: 978-988-19251-7-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)