VARIANT LOGIC META-DATA MANAGEMENT FOR MODEL DRIVEN APPLICATIONS

Allows Unlimited End User Configuration and Customisation of All Meta-data EIS Application Features

Jon Davis and Elizabeth Chang
Curtin University of Technology, Curtin Business School, Bentley, 6102, Australia

Keywords: Meta-data, Meta-model, Variant, customisation, Logic, EIS, Lifecycle, Version control, Version management.

Abstract: The scope for end users to influence the design and functionality of off the shelf Enterprise Information system (EIS) applications is usually minimal, requiring pursuing expensive vendor supported customisations. Our ongoing development of temporal meta-data EIS applications seeks to overcome these issues, through modelling rather than coding, and with the meta-data model supporting the capability for end users to define their own application logic meta-data, to supplement or replace the originating vendor's pre-defined application logic, as what we term Variant Logic. Variant Logic can be applied to any object defined in a meta-data EIS application, and can be defined by any authorised user, without the need for additional coding, and is available for immediate execution by the framework runtime engine. Variant Logic is also preserved during automated meta-data application updates.

1 INTRODUCTION

Almost every application that is in practical use is the result of hard coded program logic that has been compiled and deployed for use as part of a developer's release schedule with minimal scope for end users to influence the design and functionality of the application without expensive customisations.

Our ongoing development of a temporal meta-data framework for EIS applications seeks to overcome these 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 an associated runtime engine.

A specific objective of the framework is to also provide the capability for end users 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, to become an alternate variation of the application logic.

Variant Logic can be applied to any object defined in a meta-data EIS application whether; visual objects, logical processing objects, or as data structures. Variant Logic can be defined by any authorised user and can be executed by any user as an alternative to the original application logic, and without the need for additional coding as per the design objectives of the meta-data EIS application. The logic changes will also be available for immediate execution by the framework runtime engine, and are preserved during automated meta-data application updates that may be provided by the originating vendor or developer.

This paper reviews related works, examines the application to a meta-data based application model, and provides examples where the Variant Logic can be used effectively in real enterprises.

2 RELATED WORKS

The following related issues have guided this research.

2.1 Object Polymorphism

The polymorphism aspect of object oriented programming has provided a helpful simplification to logic localisation and streamlined coding practices (Dettmer, 1995).

An analogy in the meta-data EIS application are multiple instances of the same meta-data object each defined with different features as instances of Variant Logic.

2.2 OMG, MDA and Reusable Objects

The aim of the Object Management Group (OMG) with their Model Driven Architecture (MDA) initiative is to "separate business and application logic from underlying platform technology" (OMG, 2010).

UML is a widely adopted standard for software development but is a semi-formal language which lacks the precisely defined constructs to fully define application logic (Mostafa, 2007).

Complementary strategies aim for more plug'n'play style solutions where components are constructed to an accessible interface standard which allows the component objects to more readily interact with minimal recoding (Talevski, 2003).

In (Yan, 2009) support for multiple simultaneous versions of an application would be provided by a hybrid interim data schema evolution that satisfies both the current and proposed application functionality as a stepping stone to the final version and schema.

The temporal meta-data framework for EIS applications allows multiple duplicate objects to exist with separate logical definitions that can be selected for execution.

2.3 User Configuration

While varying in complexity (Rajkovic, 2010), the generally available configurable content for end users tends to be limited to simplistic features, with application customisation being required for more advanced features. (Hagen, 1994) identified the need for focussing on more configurable software to benefit users and developers as a joint initiative of software development to merge configuration and customisation aspects.

Any feature of the meta-data EIS application can be optionally configured by authorised users.

2.4 User Customisation

Whilst customisations may have a suitable overall business case for an organisation to take this option, it is often expensive and time consuming. (Dittrich, 2009).

Customisations to EIS systems require developers fluent in the development languages and in the detailed structure of the application logic and structure and depending on the scale can become significant software engineering exercises. (Hui, 2003) provides a model to optimise capturing the requirements for customisations.

All features of the meta-data EIS application can be defined as customised by authorised end users, aiming at the knowledgeable business user rather than the technical expert.

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 (Schmidt, 2006).

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 such as (Ortiz, 2010) who specifies alternate aspects and (Cicchetti, 2007) who identify model insertion points for code insertion. (Fabra, 2008) base their works on the UML 2 specification to seek to reduce coding and transform models of business processes into executable forms.

(Zhu, 2009) takes a strong model generation approach that modifies individual application instances for each user.

In (France, 2007) they argue for future MDE research to focus on runtime models, where these executing models can also be used to modify the models in a controlled manner. Such a direction provides not only more manageable change control, but also necessarily shifts the target of the change agent closer to the knowledgeable business end user, rather than relying solely on the technical programmer, a similar goal of our temporal meta-model framework for EIS applications (Davis, 2004).

3 VARIANT LOGIC META-DATA MANAGEMENT

Our ongoing development of a temporal meta-data framework for EIS applications seeks to remove the need for hard coding by technical developers (other than in the creation of the runtime engine and meta-data editors), and transform the responsibility of defining application logic to business analysts, knowledge engineers and even business end users.

Once the meta-data EIS application logic has been defined as the meta-data and stored as a model, without the need for additional coding, the model is used for direct execution by the runtime engine.

The model also supports a specific objective of the framework which is to provide the capability for end users 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, to become a variation of the application logic for their own specific purpose.

Variant Logic can be defined for each of the three common layers of EIS application design:

- **Database**: The definitions for data storage, management and workflow in the meta-data EIS are defined in meta-data, hence the authorised user is also able to define additional data entities and attributes that can be associated with the existing defined meta-data EIS application data.
- **User Interface**: The meta-data definitions for user interfaces can also be modified and defined with additional application features to operate with, enhance or optionally replace existing application functionality (Davis, 2005). At the personal user level (single or permitted group access) changes can be made to the application meta-data that have not been flagged as core or mandatory by the meta-data EIS application's higher level designers. Within this scope logic definers can, within their authorisation limits;
    - Remove (not delete) non-mandatory features,
    - Relocate any features between user interface locations,
    - Modify non-mandatory features,
    - Define new features to support new or existing data stores.
- **Logical Processing**: Data manipulation, workflow and processing features are provided by logical functions that are defined for processing data, similar to many of the functions in say Microsoft Excel. The functions in the meta-data EIS application are used:
    - As individual or compound functions,
    - To provide individual processing,
    - As inline or to be a user-defined function that can be used throughout the meta-data EIS application,
    - To modify the display, value of or storage of data,
    - To perform an evaluation to be used for logical workflow execution.

Variant Logic can be defined by any authorised user and can be executed by any user as an alternative to the standard application logic, subject to the defined security authorisations. The logic changes will also be available for immediate execution by the framework runtime engine.

All Variant Logic is also preserved during automated meta-data application updates that may be provided by the originating vendor or developer. Any logical conflicts that may arise during the update can be precisely identified to minimise any re-definition changes that may be required to the user meta-data customisations to avoid wholesale re-engineering efforts.

3.1 Authorisation Framework

In our meta-data EIS application model there are various levels of hierarchical authorisation that can be defined as depicted in Figure 1.

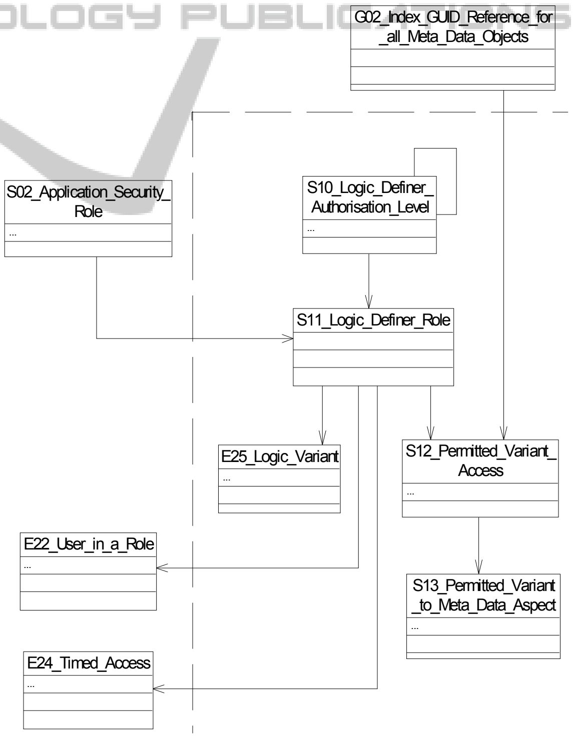
G02_Index_GUID_Reference_for_all_Meta_Data_Objects
S02_Application_Security_Role
S10_Logic_Definer_Authorisation_Level
S11_Logic_Definer_Role
E25_Logic_Variant
S12_Permitted_Variant_Access
E22_User_in_a_Role
S13_Permitted_Variant_to_Meta_Data_Aspect
E24_Timed_Access

Figure 1: Class diagram of the Logic Definer Access component of the meta-data EIS application model.

These functional authorisation processes are governed by the following principles:

- All original meta-data is created by the identified core logic definer and represents the highest level of authorisation for that meta-data

EIS application.

- Additional logic owners can be defined as (usually) lower level authorisations.
- Meta-data created 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 (where authorised), and modify undefined meta-data attributes of meta-data that was created by other logic definers where this functionality has not been restricted by the creating logic definer.
- 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 may also occur during an automated meta-data application update.

This dynamic and distributed editing feature of the meta-data EIS is drastically different from the traditional development lifecycle, further extending the provision of genuine real-time and distributed rapid application development capability, and greatly reducing the incidence of locked down or closed EIS application eco-systems that may be restricted to vendor only modification.

The Logic Definer Access (see Figure 1) uses the following classes to model the definition of the change access:

- **Logic Definer Authorisation Level**: is a simple hierarchy list of authorisation levels where a higher level of authorisation always has a higher priority and authorisation over all lower levels.
- **Logic Definer Role**: are the individual groups or roles that can be assigned to designate an identified group of functional logic definers.
- **Logic Variant**: is a designated identifier to group all of the logic changes together into a practical set as an instance of Variant Logic.
- **Permitted Variant Access**: identifies the meta-data objects that the Logic Definer Role has access to change as Variant Logic.
- **Permitted Variant to Meta Data Aspect**: identifies which aspects of the meta-data for that object can be changed as Variant Logic for that Logic Definer Role. Meta-Data Aspects are the internal groups of meta-data for each object and the effect of the change ranges from minor through to major aspects. Examples of meta-data aspect changes are: text, colours, help, sizes, position, alignment, access, type, function, assignment, validation.
- **Index GUID Reference for all Meta-Data Objects**: is a combined global register of the identifier for all instances of the defined meta-data objects (visual and non-visual).
- **Application Security Role**: is the list of the available application roles.
- **User in a Role**: is the list where Users are assigned to Application Security Roles and/or Logic Definer Roles.
- **Timed Access**: is used to define periods of allowed or denied access.

Some aspects of meta-data objects can be defined to be restricted from permitting any Variant Access, which would be applied to semantically critical application logic components.

3.2 Defined Variant Logic

All of the meta-data that defines the application logic is stored in the meta-data model, original and Variant Logic.

The key aspect that differentiates the meta-data is the identified and assigned Variant Logic for each set of additional meta-data (see Figure 2), as the Variant Logic identifies a specific group of meta-data for a specific defined purpose by its Logic Definer.

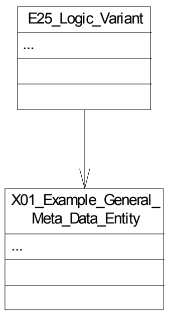

Figure 2: Class diagram of the generic assigned Logic Variant for all objects of the meta-data EIS application model.

3.3 Selection and Execution

As Variant Logic is defined, the new objects become part of the overall application pool of objects and thus are subject to the same security access mechanisms in order to provide access to the objects and how the runtime engine will manage the ongoing access to the objects.

Figure 3 provides an extract of the design for the Variant Logic Access model. It allows for assigning access to; users, roles or on an application wide basis.

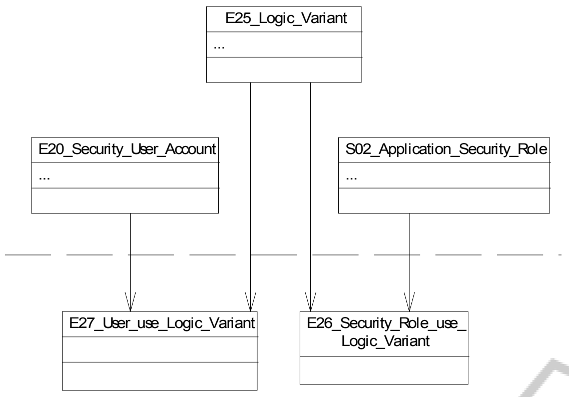
E25_Logic_Variant
...
E20_Security_User_Account
...
S02_Application_Security_Role
...
E27_User_use_Logic_Variant
E26_Security_Role_use_Logic_Variant

Figure 3: Class diagram of the Variant Logic Access component of the meta-data EIS application model.

The Variant Logic Access uses the following entities to model the definition of the alternate logic access:

- **User Use Logic Variant**: assigns individual users to access the designated Logic Variant as their new default for those objects.
- **Security Role Use Logic Variant**: assigns all users from the Security Role to access the designated Logic Variant as their new default for those objects.
- **Security User Account**: is the list of Users that are defined in the application runtime execution environment.

At runtime, the execution engine identifies available Variant Logic options for a user and selects the defined Logic Variant for execution.

3.4 Practical Examples of Variant Logic in Meta-Data Applications

To illustrate some of the operational benefits of how users can adjust a meta-data EIS application to more closely suit their local processes are, in increasing complexity:

- **Application Example**: modify the initial starting Canvas (analogous to a UI screen or form) of an application.
- **Canvas Example**: reorder the positioning and change the sizing of various Freeform Panels (logical grouping of UI object) on the screen.
- **Navigation Panel Example**: update the common Navigation Panel (analogous to menus or toolbars) with any new objects as new GUI Reports (analogous to custom reports) and Canvases.
- **Freeform Panel Example**: alter the names of several objects to more relevant local terms as well as changed the supporting text and help files. Remove data columns that are not relevant to local conditions from all Freeform Panels and from the View Tables (analogous to updateable database views).
- **Freeform Panel Example**: modify freeform entry objects GUI Object (analogous to any UI object) to become a GUI Selection (analogous to any selector such as Drop Down Box etc).
- **Freeform Panel Example**: add new View Columns (analogous to data table columns) to record unique information requirements, update validation functions to help minimise the occurrence of data entry mis-keying.
- **Function Example**: define new data columns an calculation functions to integrate new and existing data.
- **Workflow Example**: define new workflows or insert new workflow stages into existing workflows.
- **Module Example**: define a new application module including; new data objects, all UI screens, reports and workflows.

By extending the above simplistic examples with more complexity, including adding entirely new functionality, the accessibility, power and immediacy of Variant Logic becomes a key capability of the meta-data EIS application, particularly with the empowerment of a much wider base of new Logic Definers, including knowledgeable end users.

4 CONCLUSIONS

The Variant Logic capability in combination with meta-data EIS applications provides a framework that can allow any level of application customisation to be securely and flexibly applied to an existing meta-data EIS application.

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 Variant Logic capability alone will provide additional order of magnitude tangible efficiency savings due to:

- Providing an open standard of application definition (for meta-data EIS application models) that minimises the level of vendor dependence and increases the availability of competitive additional application logic definers.

- Allowing organisation business analysts and knowledgeable and power end users to directly and securely define or modify the application logic to suit local business functions, has virtually unlimited potential to offer increased efficiency due to rapid concept turnaround into concrete solutions, and reducing the high cost of using dependent vendor resources.
- When combined with the ability to merge multiple meta-data EIS applications, and their common functions progressively mapped to a single structure facilitating rapid data and application integration, without coding. (Davis, 2005)
- All Variant Logic is also preserved during automated meta-data application updates that may be provided by the originating vendor or developer. Any logical conflicts that may arise during the update can be precisely identified to minimise any re-definition changes that may be required to the user meta-data customisations to avoid wholesale re-engineering efforts.

REFERENCES

Dettmer, R., 1995. A Class Act – the Rise of Object-Oriented Technology. In IEE Review, Nov 1995, Vol 42, Iss 6, pp253-.

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

Mostafa, A., Ismall, M., El-Bolok, H., Saad, E., 2007. Toward a Formalisation of UML2.0 Metamodel using Z Specifications. In *Proceedings of 8th International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*. Jul-Aug 2007. Vol 1. Pp694-.

Yan, J., Zhang, B., 2009. Support Multi-Version Applications in SaaS via Progressive Schema Evolution. In *Proceedings of the IEEE 25th International Conference on Data Engineering*, Mar-Apr 2009, pp1717-.

Talevski, A., Chang, E., Dillon, T.S., 2003. Meta model Driven Framework for the Integration and Extension of Application Components. In *Proceedings of 9th International Workshop on Object-Oriented Real-Time Dependable Systems*.

Rajkovic, P., Jankovic, D., Stankovic, T., Tosic, V., 2010. Software Tools for rapid Development and Customisation of Medical Information Systems. In *Proceedings of 12th IEEE International Conference on e-Health Networking Applications and Services*, Jul 2010, pp119-.

Hagen, C., Brouwers, G., 1994. Reducing Software Life-Cycle Costs by Developing Configurable Software. In *Proceedings of the Aerospace and Electronics Conference*, 1994, pp1182-.

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-.

Hui, B., Liaskos, S., Mylopoulos, J., 2003. Requirements Analysis for Customisable Software: a Goals-Skills-Preferences Framework. In *Proceedings of the 11th IEEE International Requirements Engineering Conference*, Sept 2003, pp117-.

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

Ortiz, G., De Prado, A., 2010. Improving device-aware Web services and their mobile clients through an aspect-oriented, model-driven approach. In *Information and Software Technology*, Oct 2010, Vol 52, Iss 10, pp1080-1093.

Cicchetti, A., Di Ruscio, D., Di Salle, A., 2007. Software customization in model driven development of web applications. In *Proceedings of the 2007 ACM symposium on Applied computing*, ACM, New York, NY, USA, pp1025-1030.

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-.

Zhu, X., Wang, S., 2009. Software customization in model driven development of web applications. In *Proceedings of International Conference on Management and Service Science*, pp.1-4, 20-22 Sept. 2009.

France, R., Rumpe, B., 2007. Model-driven Development of Complex Software: A Research Roadmap. In *Future of Software Engineering (FOSE 2007)*. IEEE.

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.

Davis, J., Tierney, A., Chang, E., 2005. A User Adaptable User Interface Model to Support Ubiquitous User Access to EIS Style Applications. In *Proceedings of the 28th International Conference on Computer Software and Applications*, Edinburgh, Scotland, July 2005.

Davis, J., Tierney, A., Chang, E., 2005. Merging Application Models in a MDA Based Runtime Environment for Enterprise Information Systems. In *Proceedings of the 3rd International IEEE Conference on Industrial Infomatics*, Perth, Australia, August 2005.