Chapter 44
Optimized and Distributed Variant Logic for Model-Driven Applications

Jon Davis
Curtin University, Australia

Elizabeth Chang
University of New South Wales, Australia & Australian Defence Force Academy, Australia

ABSTRACT

The customization of Enterprise Information Systems (EIS) is expensive throughout its lifecycle, especially across an enterprise-wide distributed application environment. The authors' ongoing development of a temporal meta-data framework for EIS applications seeks to minimize these issues with the application model supporting the capability for end users to define their own supplemental or alternate application logic as what they term Variant Logic (VL). VL can be applied to any existing model object, defined by any authorized user, through modeling rather than coding, then executed by any user as an alternative to the original application logic. VL is also preserved during automated application updates and can also interoperate directly between similar model-based execution instances within a distributed execution environment, readily sharing the alternate logic segments. The authors also present an enhanced pre-processing architecture that optimizes the execution of Logic Variants to the same execution order of single path model logic.

INTRODUCTION

The great majority of software applications in practical use are the result of hard coded program logic that has been compiled and deployed for use as part of the developer's release schedule. Whether the result of a developer producing a commercial application for widespread release or an internal development team producing software to suit a specific internal purpose or process, the development path will follow similar traditional processes.

DOI: 10.4018/978-1-5225-3422-8.ch044

Externally developed third party software typically provides minimal scope for end users to greatly influence the design and functionality of the application – such influence is usually minor and limited to providing suggestions or advice to the developers, or via bug reporting feedback. While identified bugs in a commercial application may be a priority and receive a higher level of attention in terms of feedback from users and the response of developers it is more typical that user requests for change will suffer long periods before they are introduced into production application releases, if ever.

Expensive alternatives that are often employed by organizations that use large scale third party Enterprise Information System (EIS) or Enterprise Resource Planning (ERP) style solutions are to engage the vendor or other authorized third parties to develop specific customizations for an organization's requirements to become embedded within a new localized version of the application to support that determination.

Internally developed software may often offer some "time to delivery" opportunities in effecting new desired functionality due to a potentially higher focus on satisfying the organization's specific requirements. The overall cost effectiveness of internal development vs. the use of third party applications with customizations requires a suitable business case for each organization.

In either case, any ongoing customizations or internal development efforts over the lifecycle of the application can be a significant additional cost. An alternative option to choose to utilize commercial-off-the-shelf applications and limit modifications can minimize direct costs but impose internal organizational workflow and inefficiency costs that can also become significant and need to be identified and assessed as part of an overall business case to assist in solution decision making.

The overall lifecycle costs of maintaining an EIS style application are further compounded when accounting for the effort and costs of all major version upgrades, updates, patches and field fixes that may be released by the application vendor. These costs can be significantly magnified when the organization has employed customizations as they need to be reviewed and tested and may require re-engineering using traditional hard coding techniques during each update event to ensure compatibility. Where organizations often choose to defer or skip upgrades to reduce these update costs and any associated application downtime they still incur internal organization inefficiency costs due to delaying the uptake of the otherwise provided updated application benefits to their organization. A suitable review should be conducted to assess these effects.

Our ongoing development of a temporal meta-data framework (Davis 2004) for EIS style applications seeks to overcome these issues as an example of the model driven engineering paradigm. A meta-data EIS (MDEIS) application is fully defined and stored as a model, without the need for application coding, for direct execution by an associated runtime engine.

How do we define MDEIS applications? Firstly, we consider the class of EIS applications that we summarize as visual and interactive applications that prompt for the entry of appropriate transaction data and user events from the application users, use rules based workflow sequences and actions and utilize database transactions in a (relational) database environment to complete the actions. They are typically structurally repetitive and tend to be a technically simpler subset of possible software applications.

They generally consist of EIS and Enterprise Resource Planning (ERP) style applications such as; logistics, human resource, payroll, project costing, accounting, customer relationship management and other general database applications. The collective application design requirements are stored and available in a suitable meta-model structure and supported by an execution framework that will allow the EIS application models to be executed automatically and directly from the model, thus the transformation to the MDEIS application.