School of Information Systems

TEMPORAL META-MODEL FRAMEWORK FOR ENTERPRISE INFORMATION SYSTEMS (EIS) DEVELOPMENT

Jon Davis

This thesis is presented for the Degree of
Doctor of Philosophy
of
Curtin University

December 2014

Thesis Summary

40 years on, approximately 70% of software projects have failed [1], and today, despite continuing technology advances, enterprises are still facing the challenges that their software systems do not meet their evolving needs. This thesis addresses key issues associated with large-scale enterprise software development. It has developed a Temporal Meta-Model Framework for semi-automated Enterprise System Development, which can help drastically reduce the time and cost to develop, deploy and maintain Enterprise Information Systems.

Despite ongoing continuing technology developments, there has been no fundamental change to the outcomes and issues of large scale software production in today's Enterprise Information Systems. A recent analysis of software projects [2] by the Standish Group determined that 31% of projects are cancelled, 52.7% of projects overrun and only 16.2% they are completed on-time and on-budget.

This research project proposes that the performance of the analysis and requirements gathering, with an efficient collection of this information, can also perform the bulk of the design phase for an EIS application, largely as a simultaneous activity, with the collective design requirements stored and available in a suitable model. This research aims to develop a meta-model structure and framework that will allow EIS style applications to then be executed automatically from this model with the availability of a set of specific runtime components.

This expectation is based on the highly structured nature of Enterprise Architecture (EA) that influence the design of EIS applications which I summarise 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 utilise database transactions in a (usually) relational database environment to complete the actions. As EIS applications are typically structurally repetitive they tend to be a technically simpler subset of possible computer applications. They consist of applications such as logistics, human resource, payroll, project costing, accounting and other general database applications.

This thesis addresses user requirement and system design issues associated with large-scale software development. The main objective of this thesis is to develop an alternative development methodology by proposing a model standard for defining and

producing Enterprise Information Systems in a much cheaper and simpler way, exploring additional benefits that might be derived from subsequent usage of a model based framework.

This thesis achieves its objectives through the following solution development:

1) The definition of a model structure that will adequately model the application features required in EIS applications encompassing the user interface, business logic workflow and transaction processing capability.
2) Design accelerator mechanisms to expedite and simplify population of the model by users, with user specified model data such as rules and relationships between application objects, wizards for model data entry sequences, user interface templates, external model reverse engineering and additional model objects that will facilitate integration between multiple models.
3) Design of a prototype that would be used to automatically execute the EIS application models. This runtime engine is expected to be service based utilising any combination of technologies and deployment strategies. The high level design will document the key features and attributes of the runtime execution environment.
4) Definition of an interface language specification that could be used to access data and application services from external applications. Based on a service-oriented architecture (SOA) all functions of the solution will be available for de-centralised cloud access and integration using common standards.

Acknowledgements

I would like to thank my PhD supervisor, Professor Elizabeth Chang for her constant advice and help over the period of this thesis and for her enduring value adding comments, support and guidance required to succeed. I would also like to thank my co-supervisor Dr John Venable, thesis chair Dr Omar Hussain, and the (then) Centre Manager of DEBII (Digital Ecosystems and Business Intelligence Institute) Ms. Sonya Rosbotham for their value-adding contributions and support while completing this thesis. Their extensive knowledge, experience and advice were the key to achieving this goal. For this I am very grateful.

To Ponnie, Sazia and Vidy, so many thanks for helping me to finalise the thesis.

Whilst somewhat delayed due to the untold wonders experienced with the birth of my son Maclean, completing this work was also only made possible by the patient and loving support and ongoing trust and belief of my wife Leanne.

The acknowledgement also extends to my industry colleagues (thankyou to Andrew in particular) who assisted with review of the model designs for real world solutions, and to the staff of Newcastle University's DCITA Centre for their facilities and friendship.

List of Publications

Book Chapters

[PP1] Davis, J., Chang, E., 2014, "Optimized and Distributed Variant Logic for Model Driven Applications", in *Handbook of Research on Innovations in Systems and Software Engineering*, Editors: Dr. Vicente García Díaz , Prof. Dr. Juan Manuel Cueva Lovelle, Dr. Begoña Cristina Pelayo García-Bustelo, and Dr. Oscar Sanjuán Martínez, accepted for publishing August 2014, 45pp, IGI Global.

[PP2] Davis, J., Chang, E., 2013,"Variant Logic for Model Driven Applications", in *Advances and Applications in Model-Driven Software Engineering*, Prologue by Bran Selic, Editors: Dr. Vicente García Díaz , Prof. Dr. Juan Manuel Cueva Lovelle, Dr. Begoña Cristina Pelayo García-Bustelo, and Dr. Oscar Sanjuán Martínez, August 2013, pp1-34, IGI Global.

[PP3] Davis, J., 2013, "Runtime Integration Capability for Distributed Model-Driven Applications", in *Progressions and Innovations in Model-Driven Software Engineering*, Prologue by Jean Bézivin, Editors: Dr. Vicente García Díaz , Prof. Dr. Juan Manuel Cueva Lovelle, Dr. Begoña Cristina Pelayo García-Bustelo, and Dr. Oscar Sanjuán Martínez, June 2013, pp147-180, IGI Global.

Refereed Conference Papers

[PP4] Davis, J., Chang, E., 2013, "Distributed Runtime Capability of Meta-Data Model Based Applications", in proceedings of International Conference on Computers and their Applications (CATA-2013), Honolulu, Hawaii, 4-6 March 2013, pp295-300.

[PP5] Davis, J., Chang, E., 2011, "Variant logic automated update for MDA based enterprise information systems", in proceedings of 24th International Conference on Computer Applications in Industry and Engineering (CAINE-

2011), Honolulu, Hawaii, 16-18 November 2011, pp183-188, International Society for Computers and their Applications (ISCA).

[PP6] Davis, J., Chang, E., 2011, "Automatic application update with user customisation integration and collision detection for model driven applications", in proceedings of World Congress on Engineering and Computer Science (WCECS 2011), San Francisco, USA, 19-21 October 2011, pp1081-1086, International Association of Engineers (IAENG).

[PP7] Davis, J., Chang, E., 2011, "Lifecycle and generational application of automated updates to MDA based enterprise information systems", in proceedings of Proceedings of the 2nd International Symposium on Information and Communication Technology (SoICT 2011), Hanoi, Viet Nam, 13-14 October 2011, pp207-216, Association for Computing Machinery.

[PP8] Davis, J., Chang, E., 2011, "Temporal Meta-data Management for Model Driven Applications - Provides Full Temporal Execution Capabilities throughout the Meta-data EIS Application Lifecycle", in proceedings of 13th International Conference on Enterprise Information Systems (ICEIS 2011), Beijing, China, 8-11 June 2011, Vol 3, pp376-379, SciTePress.

[PP9] Davis, J., Chang, E., 2011, "Temporal Meta-data Management for Model Driven Applications - Provides Full Temporal Execution Capabilities throughout the Meta-data EIS Application Lifecycle", in proceedings of 13th International Conference on Enterprise Information Systems (ICEIS 2011), Beijing, China, 8-11 June 2011, Vol 3, pp376-379, SciTePress.

[PP10] Davis, J., Tierney, A., Chang, E., 2005, "Merging Application Models in a MDA Based Runtime Environment for Enterprise Information Systems", in proceedings of 3rd International IEEE Conference on Industrial Informatics (INDIN-2005), Perth, Australia, 10-12 August 2005, pp605-610, IEEE.

[PP11] 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 29th International Conference on Computer Software and

Applications (COMPSAC 2005), Edinburgh, UK, 25-28 July 2005, Vol 1, pp351-358, IEEE.

[PP12] Pudhota, L., Chang, E., Davis, J., Venable, J. 2004, "Collaborative workflow management for logistics consortium", 1st International Workshop on Computer Supported Activity Coordination (CSAC-2004) Porto, Portugal, April 2004, pp246-252.

[PP13] Davis, J., Tierney, A., Chang, E., 2004, "Meta Data Framework for Enterprise Information Systems Specification - Aiming to Reduce or Remove the Development Phase for EIS Systems", in proceedings of 6th International Conference on Enterprise Information Systems, Porto, Portugal, April 2004, Vol 3, pp451-456.

[PP14] Pudhota, L., Chang, E., Davis, J., 2004, "Modelling the dynamic relationships between workflow components", in proceedings of 6th International Conference on Enterprise Information Systems, Porto, Portugal, April 2004, Vol 3, pp495-500.

[PP15] Chang, E., Davis, J., Chalup, S.K. 2003, "A New Look At the Enterprise Information System Life Cycle - Introducing the Concept of Generational Change", in proceedings of 5th International Conference on Enterprise Information Systems (ICEIS 2003), Vol 3, pp40-50.

[PP16] Chang, E.J., Wongthongtham, P., Jayaratna, N.A., Davis, J.E., Dillon, T.S., 2003, "Ontology based solution proposal for multi-site distributed software development", in proceedings of 16th International Conference: Software and Systems Engineering and Their Applications (ICSSEA 2003) , Paris, France, 2-4 December 2003.

Statement of Authorship

Except where reference is made in the text of the thesis, this thesis contains no material published elsewhere or extracted in whole or in part from a thesis submitted for the award of any other degree or diploma

No other person's work has been used without acknowledgment in the main text of the thesis.

This thesis has not been submitted for the award of any degree or diploma in any other tertiary institution

Signed: _____

Jon Davis

Table of Contents

Table of Figures

Table of Tables

Chapter 1 - Research Motivation

1.1 Introduction

"*There's never enough time to do it right, but there's always enough time to do it over.*" – Jack Bergman [3]

"*If you haven't got the time to do it right, when will you find the time to do it over?*" – Jeffery J. Mayer [4]

With these quotes in mind this thesis is about software application development, particularly relevant to larger scale Enterprise Information Systems (EIS) applications although also with applicability to similar but smaller scale application development.

Millions of software developers around the world are actively engaged in large and small scale software developments using a huge variety of technologies and programming environments but are often developing identical or at least very similar applications.

Having worked closely in many professional software development environments over the last 30 years, I have directly observed the high degree of effort and expenditure that is consumed in continuous duplication, re-engineering and updating software applications and customisations. While always promoting and achieving a much higher degree of modelling and code re-use, and often achieving factors of

measurable productivity improvement for an organisation's software development, the key issue is that the potential magnitude of focused iterative improvements and optimisations are always destined to be constrained as there are so many inter-related and dependant technological facets to any software development.

A paradigm shift in software development is required that can truly reduce the scale of technological barriers and increase the openness of what many customers experience as a closed or locked in application environment.

To address these problems in this thesis I propose a model based approach to software application development whereby all aspects of the logical application requirements are captured in a meta-data model from which the ultimate software application is directly executed from – with no direct programming.

To further increase the effectiveness of such a paradigm it is proposed to remove one of the main technological barriers - the need for highly trained technical software programmers - and instead utilise existing business analysts, power users and even normal business users to define their requirements into the model for direct application execution.

The research overview discussed below forms the primary motivation for my research.

1.2 Enterprise Information Systems and Development Lifecycle Issues

Addressing optimisations for the development of larger scale systems for use by larger organisations can potentially address a global scale of efficiency improvements.

1.2.1 Enterprise Information Systems

Firstly, how are Enterprise Information Systems defined? [5] defines EIS as: "*The applications that constitute an enterprise's existing system for handling companywide information. These applications provide an information infrastructure for an enterprise. An enterprise information system offers a well-defined set of services to its clients. These services are exposed to clients as local or remote interfaces or both. Examples of enterprise information systems include enterprise resource planning systems, mainframe transaction processing systems, and legacy database systems*."

Similarly [6] defines EIS as "*Enterprise Information Systems (EIS) are large-scale, composite systems, consisting of software and hardware components, which should be effectively combined to ensure system efficient operation.*".

For this thesis I consider the class of EIS applications that is summarized 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 [7].

The emergence of the Internet and Cloud have provided significant opportunities for vendors with access to wider markets and for customers with greater de-centralisation options. Vendors have also largely been required to expand their development technologies in order to support the additional platforms now expected by their customer base. Previously a vendor may have only required a single development language for a specific platform – now they may also be expected to provide cross-platform alternatives to suit desktop users as well as mobile tablet and smartphone users in any location operating a variety of different platforms.

1.2.2 Distributed Enterprise Information Systems (DEIS)

A later extension to this thesis was consideration of Distributed Enterprise Information Systems (DEIS) whereby large geographically de-centralised organisations utilise multiple instances of similar EIS applications to service the potentially differing needs of remote and regional business units.

Whether the DEIS instances are implemented in the cloud, or as discrete instances perhaps serving customers based on the limitations of regional communications links, they pose similar integration problems between the DEIS instances.

Typical major problems with larger decentralised organisations is the integration and transfer of data between business units, including the progressive processing and rollup of data between hierarchical business levels. When business units also utilise

different EIS applications the data sharing can be more problematic due to the need for additional business logic verification.

1.2.3 Software Engineering in the last 40 years

In addition to what has often become a huge increase in the complexity of software development and integration is the fundamental performance (problems) of many software development projects as evidenced by this brief excerpt of software project performance studies:

- 50% of projects failed completely plus another 40% classified as partially completed [8].
- 55% of 1,500 project managers surveyed in the UK indicated their projects exceeded budgets [9].
- Only 29% of projects reported in 2004 by the Standish Group were completed successfully [10].
- Approximately 70% of projects failed and would continue to fail [11].

Whilst this thesis does not directly address the project management issues of software development projects, it does need to acknowledge that the additional complexity and platform burdens placed on software developers will be reflected in the associated expense and delays of poorly executed projects – as a contributing factor to the ongoing expense of traditional software development.

1.2.4 Software Development Life Cycles

The traditional project management lifecycle has typically followed variants of the phases of; establish requirements, create a design, build to specification, test conformance and finally deploy the solution.

As with any engineering or construction project, computer and information systems are most sensibly developed by adhering to formal methodologies that are specifically designed for their optimisation and management.

Waterfall: was an early methodology derived from existing standard engineering manufacturing processes [12] and is based on a linear or sequential series of phases. Work in a phase must be complete before proceeding to the next phase. Accordingly the methodology is well suited to a fixed scope [13].

B-Model: extended the Waterfall model to include ongoing lifecycle software improvements as a logical continuation of the original development processes to allow for software evolution [14].

Spiral: combines the sequential discipline of the Waterfall model with an incremental or prototyping approach and is often used for high risk projects. It is based on successive loops through four phases or quadrants; Objectives, Risk Analysis, Development, and Planning [15].

V-Model: developed by NASA in 1991 [16] as a variation of the Waterfall model, the V-model attempts to overcome perceived limitations of the Waterfall model by ensuring testing (and therefore problem detection) occurs earlier in the lifecycle.

Unified Process Model: is use case driven and iterative and uses models defined using the Unified Modelling Language (UML) [17], such as use-case, activity, class, object, interaction and state diagrams providing a functional view obtained by modelling expected user interactions, making it particularly suitable for business logic and front-end applications.

Agile: refers to a multitude of software development methods based on iterative and incremental development, where requirements and solutions evolve through collaboration between cross-functional teams. It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change [18]. Agile is recommended when exacting requirements are unknown or difficult to elicit.

Scrum: is a form of Agile development based on defined "sprints" with the aim for the assigned team to create a shippable product increment, from the product "backlog", the prioritised requirements. The "sweet spot" for Scrum style projects involves "a small, co-located team; an on-site or available customer representative; an emphasis on coding and testing early; and frequent feedback into updated requirements".

Extreme Programming (XP): promotes high customer involvement, rapid feedback loops, continuous testing, continuous planning, and close teamwork to deliver working software at very frequent intervals. It is based on 12 supporting practices of; planning, small releases, customer acceptance tests, simple design, pair programming, test-driven development, refactoring, continuous integration, collective code ownership, coding standards, metaphor, sustainable pace. The customer works

very closely with the development team to define and prioritize granular units of functionality referred to as user stories [19].

Dynamic Systems Development Method (DSDM): is an agile delivery framework built on 8 working principles supporting five phases; Feasibility, Foundations, Exploration, Engineering and Deployment. It is recommended for creating solutions that are required quickly. It has been formally documented and is freely available for use [20] as well as endorsed by a leading project management methodology organisation [21].

To an extent each of these methodologies are fundamentally waterfall in nature, although each with a differing level of focus. Whether they are iterative or incremental they still tend to necessarily follow the same general set of phases in determining requirements, planning and priorities, building, testing and ultimately deployment – some work in smaller blocks and more frequent cycles but at the end of any software development project run by different software development methodologies the output would largely be expected to be similar in terms of functionality as a similar solution to a problem would likely be developed.

Accordingly as a similar magnitude of work needs to be done, project timeframes and expense may vary but to a degree but you wouldn't expect major variations. Of course there will always be variations introduced by more experienced teams and project managers and some methodologies will better suit some teams and organisations than others but overall the average or expected variations would tend to be within a common range over time.

Is this good enough? Clearly any incremental development savings are worthwhile for an organisation and when adopted by many organisations the overall benefits are increased but overall it is still only incremental improvements for which I feel that better opportunities may exist.

1.2.5 Automated and Semi-Automated Software Engineering

A long-time term of derision in programming in the information age is the "CRUD" – standing for Create, Read or Retrieve, Update and Delete [22]. It refers to the continual repetition of program code that is required for database operations, where every database table and transaction requires similar code to be produced to ensure that all database transactions can be processed securely.

CRUD can also be considered a metaphor for all of the repetitive coding that occurs in software development. Consider the efforts of millions of computer programmers around the world, often working on similar applications, in different or identical coding languages and platforms. There are massive efforts being expended on repetitive and duplicated software coding.

Integrated Development Environment (IDE): The most fundamental software coding is where all software is developed using the basic IDE provided by a chosen vendor of the core software development tools. Such an environment will typically only provide the editor, compiler and debugger to allow software programs to be created. Using only an IDE, programmers must develop all code from scratch.

Third Party Libraries: Commercial options or extensions to the IDE or development environment are often available that provide pre-coded functionality as coding accelerants – these may be provided by the original vendor or by specialist third party library developers that aim to provide substantial additional capabilities via their pre-developed extended functionality libraries providing faster or higher quality development.

Open Source: The Internet age has also given rise to an even greater availability of coding shortcuts. Many organisations and individuals publicly make available their software source code and libraries for general use, often royalty free. Regardless of whether the individual open source providers have based their reasons on political, social, ego, marketing or financial motivations, there is a massive availability of free or minimal cost software available from the open source community ranging up to fully commercial quality high feature components.

The sensible use of pre-built or readily modifiable software components has the potential to both increase the functionality and reduce the development effort of software projects.

Rapid Application Development (RAD): can be defined as "any software life-cycle designed to give faster development and better results and to take maximum advantage of recent advances in development software"[23]. The basis of RAD was to counter the major problems of Waterfall style methodologies - that large applications took so long to build that the requirements may have changed enough to render the solution inadequate or unusable.

Computer Aided Software Engineering (CASE): can be defined as "the use of a computer-assisted method to organize and control the development of software,

especially on large, complex projects involving many software components and people. Using CASE allows designers, code writers, testers, planners, and managers to share a common view of where a project stands at each stage of development" [24]. CASE tools can consist of repositories and accelerants for; capturing requirements, design specifications, source code, use cases, and for code generation. By capturing and managing the CASE tool's knowledge domain in a structured format the CASE tool is also available as a multi-user collaboration environment.

Model Driven Design (MDD): refers to toolsets and development environments that first capture the application design requirements into a model and then generate the appropriate source code for compilation to the final application executable. Many RAD and CASE tools provide examples of these toolsets. Within the feature scope and presentation layouts supported by the toolsets, specific application subsets can be efficiently developed with these tools for a variety of local execution and remote Internet environments.

A vexing problem in RAD, CASE and MDD tools and modelling and generation in general is that the functionality of the generated applications is usually limited by the scope of the available functionality supported by the toolset. Best case solutions to allowing modifications to the generated output involve allowing direct modification of the generated source code, or allowing for embedding external objects or links to external code to provide the required features that cannot be provided by the modelling toolset.

The former case can readily incur the additional problem of loss of synchronisation between the base model and the modified output resulting in continual re-work or progressive logic mismatches. The latter case can be plagued by restrictions on the structural object insertion points combined with the associated issues of inter-acting with or counter-acting any undesired effects of the often fixed generated logical structures.

Restrictions on the take-up of many RAD and CASE tools often include the initial costs to procure the toolsets, plus the efforts to both establish the toolset knowledge and expertise in development staff as well as integrating the toolsets into the overall local development environments and methodologies. When multiplied by an often large number of development staff the initial costs can be prohibitive for many organisations. Additional internal organisational bias can be generated by

overall “resistance to change” as well as any legitimate or perceived issues with inherent limitations of the toolsets.

[25] investigated the low adoption rates of CASE tools citing rates over 50% where tools where neither understood nor used at all, with partial adoption rates below 30% i.e. where a tool was in frequent use that assisted with part of the development lifecycle. The most popular toolset cited by far was a simple diagrammatic documentation tool.

[26] examined the factors influencing software development effort and concluded that “a majority of organisations reported that CASE has not brought about any change in productivity” primarily due to “the fact that CASE tools tend to support the old way of developing software”. In contrast they concluded that “the use of RAD can significantly reduce development effort” due to “the short time between design and implementation often means the system is much closer to the needs which constantly evolve during the development process” noting RAD’s particular usefulness for “for projects where the scope is small or work can be broken down into manageable segments”. Such key problems in adoption and integration occur particularly when only addressing partial aspect of the development lifecycle while maintaining legacy processes for the remainder.

The typically lower cost of procuring third party software libraries has contributed to their widespread adoption. While typically providing only opportunity optimisations rather than the potentially widespread effects of RAD or CASE tools, most developers have adopted at least some third party software accelerants.

Notwithstanding the globally attained benefits of most developers obtaining good efficiencies from their use of third party software, and some developers achieving excellent benefits from RAD or CASE tools, the overall development landscape has not fundamentally altered. Perhaps the overall level of CRUD has been reduced but consider the efforts of millions of computer programmers around the world, still mainly working on similar applications, in different or identical coding languages and platforms. All that surplus software coding effort - the duplication, re-engineering and updating - still has not yet been fundamentally simplified or reduced.

When combined with the relative explosion in recent years in the number of additional platforms and technologies, particularly in support of mobile computing, the need for more fundamental optimisations and ubiquitous multi-platform solutions is clear.

1.2.6 Key Challenges in Software Engineering - Alignment of IT to Business Objectives

A perennial concern of business is to ensure that the too often too technically focussed IT (Information Technology), IS (Information Systems) or MIS (Management Information Systems) groups more closely support the business objectives of the organisation rather than concentrating on purely technical issues.

An IT group needs to balance its technical support responsibilities with the need to help direct the business towards technological solutions that optimise or improve their business processes. Achieving a suitable balance can be a difficult problem to solve in an organisation particularly where the IT functions may still be maintaining a traditional focus on the technical aspects of IT support.

A traditionally developed software application requires a great deal of specialist assistance during the development lifecycle. Business Analysts (BA) are required to assist with requirements elicitation during an Analysis phase. Technical architects are then required during a Design phase to translate those requirements into a suitable system design. Programmers, report writers, documenters are required to develop the system components during a Build phase and then assist testers during the subsequent Test phase. Additional training and technical staff are then required to implement the solution during a final Deployment phase. Progressive iterations of similar but usually smaller lifecycles will be required for any subsequent enhancements or upgrades. The overall large labour intensive costs of EIS style applications can be a prohibitive factor limiting the scope of many business solution projects.

Extensive technical IT resources are required throughout these phases whether as individual or multi-skilled specialists. The organisation's Subject Matter Experts (SME) will be utilised primarily during requirements gathering where the collated requirements will then be agreed and passed onto further IT specialists for ultimate development. As the role of most IT specialists tends to focus more on technical expertise than on business processes there is a progressive risk through the development lifecycle where the progressive translation of the original business requirements into design architecture and components specification then into program code, reports and data then back into user guides and training is subject to degrees of iterative obfuscation through reduced communication clarity, technical layer translation losses, and functional limitations of the technical solutions.

Where major new EIS applications, modules or enhancements are required the onward march of time can also severely hinder the ultimate practicality of any solution delivery as extended project development periods of many months or even years can invalidate major aspects of a delivered solution. During the development periods the organisation's business processes may have continued to evolve extensively resulting in potentially greatly reduced benefit realisation from the development project.

A growth in the use of more Agile based project management methodologies is aimed at trying to reduce some of these effects however these methodologies mainly assist with improving the alignment between the business processes and the IT specialists. The heavy reliance on specialist IT resources remains, along with largely similar overall levels of development effort.

1.2.7 Maintaining Quality Standards and Minimising Integration Issues

The natural evolution of an organisation considering its growth and available capital and associated progressive procurement of business solutions almost exclusively results in a disparate solution environment. Very few organisations start with and continue the means to always maintain a consistent corporate IT business solution that is also eternally expandable.

This results in most organisations' solution environments comprised of multiple component systems, often utilising different technology architectures, requiring various forms of data and process integration to have been developed and maintained. The disparate solution components also tends to disrupt the efficiency of many business processes as the processes themselves may have evolved over time but the implementation of the solution components naturally tends to lag the requirements as well as rarely fulfilling an exact match of all requirements either at initial implementation or throughout their useful lifecycles.

Thus the fulfilment of the business processes tends to progressively become a mash of manual and automated segments of business solution components loosely coupled with manual or integrated data transfers. When a particular business process solution finally becomes too painful and expensive to put up with it, it becomes the subject of an optimisation or re-development project.

Maintaining this variety of business system components, each with its own look and feel, and set of operational sub-processes is not just an interaction and consistency issue for users who need to interact with multiple systems. IT groups have to maintain specialist support for each disparate system that constitutes the organisational business system pool plus establish and maintain any required automated integrations between the systems – these integrations can be up to an order of $O(N^2)$ for N systems [27] when all of the systems are required to provide some degree of integration.

Minimising the diversity of systems while maintaining a suitable mix of best of breed solutions is a clear optimisation to benefit any organisation.

1.2.8 Reducing Ongoing Maintenance Lifecycle Costs

Establishing a new EIS application is simply the start of a long process of continuing patches, updates, customisations and platform upgrades that will often ultimately exceed the initial implementation costs. Initial costs may often represent only 20% of the final lifecycle costs [13].

Purchasing a Commercial Off-the-Shelf (COTS) product will rarely satisfy an organisation's precise requirements, as products must satisfy a maximal common subset of potential features, often resulting in a wide gap between the requirements of the organisation and the benefits provided by the solution. [28] summarise this perfectly as "Customers may have to compromise on requirements not satisfied by any available product or request products modifications" and provide an analysis of the problems and risks arising from considering a COTS-based solution.

Selecting a COTS product will often result in varied combinations of; the organisation commissioning or developing specific customisations to provide the missing features, or accepting the feature omissions and modifying their business practices. Both options tend to cause additional cost and pain to an organisation with the former case referred to as the required glueware by [29] in their case study. A simple example would be a non-configurable system that uses say, an incrementing numeric Job Number field to identify different tasks or jobs in an organisation. Perhaps an organisation may currently be organised with multiple departments that use a composite key of the form XXXNNNNN where the XXX identifies jobs allocated to specific internal departments, and NNNNN is their own numeric identifier. To adopt such a COTS solution would under either of the alternatives require the organisation to; develop or commission suitable customisations or

modifications to provide the required original requirement (which could potentially be a major cost), or alternately, modify their internal processes to match the provided COTS functionality, which may also require additional process and other systems modifications, re-training, re-printing etc.

Multiply the above example by potentially hundreds of small and/or large COTS products deficiencies for each customer, and maintained over the products lifecycle, and the lost opportunity costs escalate rapidly.

EIS application developers have to provide a high level of feature availability in their software products to attract and maintain customers but through commercial reality need to maximise the availability of the most common requirements over less used options. Using traditional development means, neither is it practicable for them to allow ultimate flexibility in their products due to technology limitations nor necessarily desirable as they need to maximise retention of their long term intellectual property.

Another common issue with customisations is that their functionality often requires re-engineering whenever the original software developer releases patches or minor or major updates. As the official updates are generally warranted to operate only with the unmodified application, every customisation needs to be individually reviewed and tested for compatibility and potentially re-engineered to maintain compatibility occupying the greatest proportion of the upgrade team's software developers and business analysts, up to 80% of their efforts [30].

An unfortunate side effect of this ongoing customisation management process is that many organisations choose to not apply patches, updates or upgrades as they become available due to these and other implementation and compatibility costs. The organisations may skip many minor or major improvements and rather adopt the occasional big-bang approach and only upgrade very periodically. Some choose to only upgrade when their current application version is threatened with or occasioned by loss of support by their application vendor.

An often unconsidered effect of these update postponements is that the organisation is missing out on many application bug-fixes as well a myriad of new functionality that otherwise could reduce operational problems and increase efficiencies. As stated in [31] "Most business and IT execs put off upgrades as long as possible to avoid costs and minimize business disruption" – only 5-10% of enterprise

software customers move to the latest release, with 40-50% on the prior release, with up to 10-20% on releases no longer supported by the vendors.

Until more economical upgrades and a greater level of application flexibility is available, organisations will continue to suffer a very high level of ongoing effort and cost to both maintain and add functionality to their EIS applications.

1.3 Benefits of Round Trip Modelling and Application Generation

It is my profound belief, and the subject of this thesis, that a particular style of Model Driven Engineering (MDE) could provide the major efficiencies to help significantly reduce this effort, and potentially by orders of magnitude if adopted on a global scale.

MDE can be defined as "*a paradigm that makes use of models as basic artifacts in the software development process*" [32], that is, abstract representations of the knowledge and activities that govern a particular application domain.

By effectively removing all coding from the development of EIS style applications and instead replacing this effort with efficiently specifying an EIS application model there is an opportunity to make major savings. If the model were capable of capturing the full logical application definition, and the application could then be executed directly from the model then the only coding efforts required would be in developing and maintaining the modelling capture and execution environment.

If such a solution was execution feature rich and efficient in terms of capturing the logical specification and globally applicable to a large proportion of EIS customers then it may be a possible solution to thus achieve significant factors if not orders of magnitude of savings by removing (or at least greatly reducing) much of the duplicated coding efforts that currently occur. In [33] we demonstrated savings potential of up to 80% in the lifecycle development of an EIS product with similar potential for software costs to be passed onto customers by such a model-based EIS vendor. Far greater savings and efficiency opportunities can be offered by efficiently offering greater user configuration options (in place of what are currently considered application customisations) to modify EIS applications themselves – this is a feature

that I refer to as Variant Logic (VL) (which will be discussed in greater detail throughout this thesis).

Where might these potential savings be made?

A large EIS application or indeed any similar application must necessarily start with a review of its requirements and the preparation of a design. This thesis will propose that performance of that analysis (which always needs to be documented or captured in an appropriate format) combined with an efficient collection of this information (as a capture of the requirements directly into an efficient model structure) can also perform the bulk of the design phase, largely as a simultaneous activity. Hence these two steps may be merged in our proposed lifecycle [34].

Further, with the collective design requirements stored and available in a suitable model format (from the first combined stage above), I believe that most (if not all) EIS style applications could be executed automatically with the availability of suitable runtime components. This expectation is based on the well-structured nature of EIS applications; highly visual and interactive applications that prompt for the entry of appropriate data by the application users, employ strong rules based actions, and utilise database transactions to complete the action.

The introduction of such an approach has the potential to drastically reduce the time to develop and deploy an EIS system. Effectively, once the analysis and design have been completed the system would become available for immediate use! The virtual elimination of the coding, combined with the minimisation of the testing and deployment stages has significant benefits for both the developer and the end users.

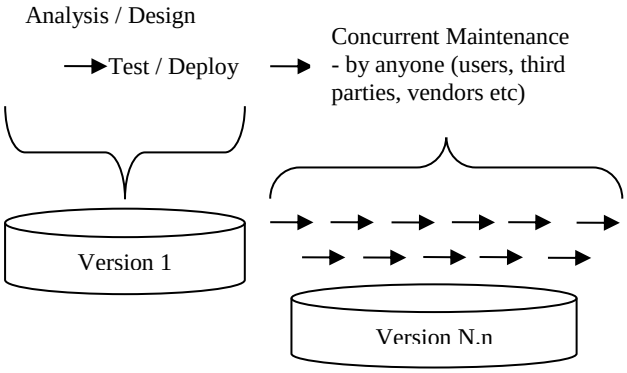


Figure 1 – Model Based Development Methodology (MBDM)

The methodology depicted above in Figure 1 differs from the traditional software development lifecycle which typically proceeds along some variant of the phases of; Specification or Requirements Analysis, Design, Implementation, Testing and Deployment with the project then entering a Maintenance phase which would also usually consist of similar lifecycles [35]. Figure 2 depicts this traditional lifecycle.

In this Model Based Development Methodology the first two traditional phases of Requirements Analysis and Design could largely be combined into a single phase as the capture and specification of the requirements into a suitable model format – such a model format is one of the primary aims of this thesis. This application model would also by its nature capture and infer the majority of any application design features as an abstraction for the logic definers of typical application features without the need to delve into complex application code.

If a suitably simple and intuitive but feature rich design and editor environment is provided to capture the application requirements it is a further objective that knowledgeable business users could be provided with these tools to effectively define (and thus create) the EIS applications themselves. This fulfilled objective could further greatly reduce the time and cost of developing EIS applications by avoiding the arduous and expensive current methods of translating business requirements into technical specifications, and utilising often large teams of highly skilled technical programmers to develop and build the software. Designing such editor components and wizard based design accelerants is also a primary aim of this thesis.

The traditional Implementation phase would fundamentally be eliminated as the supporting model's framework and runtime execution environment would directly execute the captured defined application model with no further or specific coding required. Specifying the execution framework prototype requirements is another primary aim of this thesis.

The traditional Test phase would still be recommended (although potentially optional with model, editor and runtime execution maturity) as part of the MBDM however it would be significantly reduced in scope as only the modelled semantic logic would need to be tested, fundamentally excluding the need to also test all other aspects of the usual complete base of code syntax, as common runtime objects and modules would provide the underlying services. A minor Deployment phase would also be combined at the end of testing as once application access has been initially provided to end-users, any subsequent model based application updates would occur

automatically through the runtime execution framework, removing the often significant manual and semi-automated traditional update processes.

The final ongoing Concurrent Maintenance lifecycles of the MBDM represent similar ongoing lifecycles as a traditional lifecycle except for two main differences; firstly, each lifecycle would be a correspondingly smaller (in terms of effort) MBDM lifecycle than a traditional one, and secondly, the MBDM will offer that many additional very low cost enhancement lifecycles where application users can define their own personalised application logic changes, can occur as Variant Logic (discussed in greater detail later in this thesis).

This thesis aims to investigate further options and benefits of model based development and develop an alternative development methodology using a model standard that can be extended upon for defining and producing Enterprise Information System style applications.

1.4 Challenges in Application Modelling and Generation

The majority of CASE and RAD toolsets concentrate on modelling only a portion of an existing development lifecycle. While the toolset may produce ready benefits in the management or development of that aspect of the lifecycle it may only provide minimal advantage for other lifecycle processes.

The search for a true and full application modelling and generation environment could be software developments' "Holy Grail". If the efficient capture of requirements and generation of applications can be effectively achieved without the need for relying on a lot of time and software programmers then there will be a revolution in the definition of cheap EIS software and a lot of redeployed programmers. Perhaps the fact that no organisation has yet claimed such a dominating market position is the best evidence that this objective has not yet been reached.

Even this thesis is only proposing a candidate solution for a particular class of software - for EIS style applications that can be considered to be a technically simpler subset of software solution. However, if such an initial objective can be attained then the confidence to extend the model and generation capabilities to other more complex software application domains can be gained.

A primary challenge is to ensure a suitably simple design metaphor that will readily promote the capture of the business application requirements. The simplicity

of this model editor is required to address its use towards non-technical users – the business analysts and power users within an organisation and even normal users for simpler requirements and changes. By removing the need for technical programmers the aim will be to produce the modelled application much more speedily and economically.

The current suites of visual form designers, database designers and workflow modelling tools form a solid starting place for such an editor when combined to operate to a common model.

However, while macro level tools can provide a high level approach to defining or generating the bulk of the higher level functionality, it is always the lower level processing details that form the ultimate system glue in tying in all of the major components into a suitable set of logical workflows. This aspect is also often a major impediment to wide adoption where these non-visual components require coding in the underlying native codebase to perform the additional processing logic.

There is a largely untapped huge worldwide knowledge base of business aware power users that are adept in business spreadsheet use. These users are already familiar with the Functions that these business spreadsheet use to perform logic processing. By using similar Functions to provide all of the non-visual programmatic features, in combination with the common visual aids, could open up the domain of EIS application design and development (or modelling as proposed here) directly to those who already know the business requirements, to directly produce the solution they require.

Assuming that the model correctly captures the application requirements, the remaining primary challenge is to execute the model directly without the need for additional code generation or modification. Here it is not of direct concern whether intermediate code generation occurs as long as the EIS application definers are able to execute the modelled EIS application directly.

That said, there are significant differences in; defining a model and then waiting hours for some form of automated compilation and deployment to complete, compared to making a change to the model and immediately executing the application model.

The former case above would certainly satisfy the overall objective by adequately allowing non-technical staff to effectively design and generate EIS application and accordingly generate the scale of benefits expected. Although the implication that

larger scale model would thus require larger application generation efforts and times would hinder model definition continuity without an additional focus on a current model definition session that provided immediate or fairly short-term feedback.

Thus, there is a definite preference for a more dynamic solution. The latter more dynamic case above is far more preferable as it provides for a more interactive and iterative definition or design scenario for all concurrent model definers. By utilising either direct execution from the model or via a form of Just-In Time compilation (JIT) all model definition changes (to the application definition) whether minor or major, cosmetic or structural, would be reviewed with close to immediate feedback.

How realistic are expectations for the potential solutions to these challenges?

Models to capture the requirements of applications, and not just EIS applications, have been in existence since the first days of computing. Every discrete programming language is its own specification for an application model to be defined using its specified source code. The modern terminology for these conceptual models is an ontology. The proposed model definition will be an example of such an ontology for the EIS application domain, comprising of high level aspects that provide major or macro level functionality but also including lower level or atomic statement functionality to provide for the finest detail logical processing.

A model editor or application designer can largely be composed of objects similar to the major components of current visual IDEs. As the target audience consists primarily of non-technical users, although nothing precludes technical users from being expert users, the binding logic will be based on Function definitions, many similar to the already commonly used spreadsheet variants. Much of this functionality already exists in various proven formats.

JIT technology has also been in various use since the origins of computing although the term interpretation or an interpreter may perhaps be more appropriate in this instance as the key requirement is for the currently used and requested application model segments to be interpreted into an appropriately executed application segment. JIT technology is however an appropriate metaphor to upgrade the classical distinction of an interpreted solution as necessarily of poorer performance.

Popular long established JIT technologies such as Java have demonstrated the suitability of application development based in such technology. However, there is the potential for individual model execution requests to involve the request and processing of hundreds or even thousands of other dependent sub-model components,

requiring a high level of optimisation of such dynamic processing of the model logic components.

1.5 Research Objectives

The key objective of this thesis is to define the major elements of a new framework for developing a solution for the preceding social and economic issues that are affecting EIS application development in business and industry.

Generally, EIS application acquisition in business and industry involves one of two approaches. The first is to purchase an EIS application and modify it to suit the business operations. Often it requires the company to change its business operations to suit the needs of the EIS application as customisation of the software is very expensive. The second approach is for a custom-made EIS application to be developed which is tailored to their business operations, if an organization determines that this approach is more economical than the cost of the purchase and modification option. Each vertical business or industry operates with slightly different methods and procedures and while there are well defined ISO (International Organisation for Standardisation) [36] standards for some operations areas, as well as accounting and human resource management, customisation can still be performed if the company operates quite differently.

A method is required to develop and modify EIS application functionality with less expense and time and with a reduced pressure to alter existing business processes. This research is trying to develop a framework with which to solve such issues which will provide a generalised operational software framework for EIS applications with customisable features.

The fundamental aim of this research project is to develop a meta-model structure and framework for Enterprise Information Systems that can be used as the sole source for automated execution of the modelled application by interpretation of the model data, in conjunction with user events, the modelled workflow sequences and actions, and the defined database transactions. There are 4 primary outcomes expected:

1. The definition of a model structure that will adequately model the application features required in EIS applications encompassing the user interface, business logic workflow and transaction processing capability.

Most design and modelling applications specialise on a specific tier or layer of the design - by capturing all of the required model attributes within a single model allows the entire application to be considered. By initially considering EIS applications which are a simplified subset of applications then a more realisable target is set to be achieved by utilising a simpler model than that of groups such as the OMG-MDA (Object Management Group – Model Driven Architecture) organisation which tend to target highly technical development staff rather than business users.

2. Design accelerator mechanisms to expedite and simplify population of the model by users, with user specified model data such as rules and relationships between application objects, wizards for model data entry sequences, user interface templates, external model reverse engineering and additional model objects that will facilitate integration between multiple models.

These accelerator constructs will be dependent on the final structures and workflows within the model although some aspects will be similar to other widely available reverse engineering functions. A significant feature that will be considered is the issue of merging different application models which can be achieved by specifying nodes of commonality between the models and automatically executing the combined application model – such a simplified system of merging, sharing and integrating disparate applications is expected to provide significant benefits by reducing data duplication and workplace repetition – analogous to a simple method of integrating existing disparate EIS applications this is a particularly unique and advantageous feature of the framework.

3. Design of a prototype that would be used to automatically execute the EIS application models. This runtime engine is expected to be service based utilising any combination of technologies and deployment strategies. The high level design will document the key features and attributes of the runtime execution environment.

4. Definition of an interface language specification that could be used to access data and application services from external applications. Based on a Service-Oriented Architecture (SOA) all functions of the solution will be available for de-centralised cloud access and integration using common standards.

As a direct outcome of the model structure this standardised interface specification, likely based on a web services implementation, would provide complete abstraction from the underlying physical database instances and structures, as well as abstraction from the runtime components, and (when implemented) provide full

access to all allowable and available data and application features from any external systems that support the (to be assumed) web services interface. Such an interface would provide completely platform independent data and application feature access from the model-based EIS applications to other external applications – features that are not commonly available in current EIS applications.

1.6 Research Scope

In order to drastically reduce the net effort in developing, customising and maintain the described subset of EIS style applications, this thesis presents a meta-data based application model framework. The proposed framework forms the basis of an application development solution that is targeted at non-technical business analysts and power users rather than the existing reliance on software programmers.

EIS application development effort can be increased by many associated issues such as project management methodology, technological toolsets, programmer expertise, system development lifecycle adherence and business engagement. While touching on some of these aspects, particularly on where additional benefits might be achieved, this thesis focuses solely on the technological aspects of developing a meta-data based application model framework and the associated operational flexibility issues associated with such a solution.

This framework aims to open up the area of EIS application development by replacing its current processes with a simpler methodology of directly involving and enabling knowledgeable business users to directly define their requirements into an executable application model repository.

1.7 Plan of the Thesis

The thesis is structured as follows:

Chapter 1 - Research Motivation introduces the issues and motivation behind EIS application modelling and identifies the research objective and scope.

Chapter 2 - Existing Work discusses the current related research and development in the areas relating to EIS style application modelling and generation.

Chapter 3 - Problem Definition clarifies the style of EIS applications that are our problem domain, analyses the application development issues considered in the

literature review, clarifies the research problem and proposes the basis of the model based solution.

Chapter 4 - Conceptual Framework for Temporal Meta-Model for Enterprise Information Systems outlines the proposed conceptual solution in response to the research issues mentioned in chapter 3. This chapter gives a definition of the concepts used in this thesis and presents the requirements of EIS applications and how they are met by a meta-data based EIS application model. It also lists the additional features that would be provided by this framework solution. The framework promotes strong advantages in customisation, merging applications, and temporal and distributed execution.

Chapter 5 - Instant Interaction EIS System Modeller provides an in-depth analysis of the major model elements of the proposed meta-data application model structure and how they support the fundamental requirements of EIS applications plus the advanced features provided by the model based approach. The chapter includes summary model excerpts from the master CASE design used to capture the meta-data based EIS application model specification – more detailed excerpts are available in the Appendices.

Chapter 6 - Agile Platform for Dynamic Systems Change Management reviews the options and design considerations for the runtime execution environments of the meta-data based EIS application framework. Various architecture options are considered with emphasis on the advanced functionalities that are required to be supported such as temporal and distributed execution, model element version control and automated updating, variant logic customisations and security for logic definers..

Chapter 7 - Accelerants for the Iterative Design of EIS Models presents options for how the creation and editing of the models can be achieved. From options with a basic model editor, to reverse engineering, I consider various additional accelerants unique to the model based approach such as model merging and an iterative self-defining editor..

Chapter 8 - Universal Access to Temporal Meta-Data Framework for EIS in the Cloud covers the basic structure of the internal command structures of the meta-data based EIS application framework with emphasis on the Functions syntax that provide access to the logical features of the framework, as well as the model element object addressing. The supporting web services can provide access to all model elements as the primary remote access mechanism for external systems access..

Chapter 9 - Research Validation – Case Studies for Meta-model Framework reviews basic and advanced examples of how the meta-data based EIS application framework can be used to model and subsequently execute EIS applications. A full example working of a commonly available order processing application is used as the primary basis to demonstrate the applicability of the general feature set to real work applications..

Chapter 10 - Conclusions and Future Directions concludes the thesis by reviewing the applicability of the basic and advanced features of the framework, the achievements of this dissertation, the benefits of the approach and recommendations for future work.

The **Appendices** provide additional references to materials developed during this thesis such as; information on the detailed master CASE design used to capture the meta-data based EIS application model specification (the full models are available in the supplementary materials), published papers, a glossary of terms and bibliography.

1.8 Conclusion

This chapter explains how software development, particularly focusing on EIS style application software development in this thesis, as a globally large, complex and expensive effort has not generally received the magnitude of benefits expected by the variety of project management methodologies and systems development lifecycles and methodologies that have been attempted. It is our expectation that a fundamental paradigm shift is required, to progress to model based application development with a greater emphasis away from technical programmers towards empowering business users with an application logic defining capability.

In the next chapter, I provide an overview of the thesis literature survey with further review of the current state of project management lifecycles, technologies, software development methodologies, and application modelling and generation technologies and tools.

Chapter 2 - Existing Work

2.1 Introduction

In this chapter, I provide an overview of the thesis literature survey with an evaluation of the current state of project management lifecycles, technologies, software development methodologies, and application modelling and generation technologies and tools, to review their effects on reducing the lifecycle effort involved in software production.

The vast majority of software is developed using what I term as traditional application development – where the majority of the source code is produced manually by technical software programmers. The programmers will usually utilise other in-house or third party software libraries to reduce the duplication of some coding but in the main the software will be designed, coded and (at least initially) tested by technical software programmers, under the umbrella of an appropriate project management regime.

There are many advanced modelling toolsets that allow some aspects of the application requirements to be modelled and the software can then be automatically generated to some degree. Some toolsets will even generate the majority of the application code. However, across industry the usage of these tools is in the minority.

A further important issue is that too often the business subject matter experts are kept at “arms reach” too often throughout an often lengthy software development

process resulting in outcomes that have inadequately captured requirements, or mis-translated requirements into incorrect solutions, or perhaps the final solution simply took "too long" and now the requirements have evolved in a fast paced and competitive world. Subject matter experts need to be more involved in producing software application solutions and development time and effort needs to be reduced, requiring more fundamental changes in the way application development is undertaken.

There has been much research and industry focus on developing models that can be used to automatically generate parts or all of the source code for applications, and permitting further source code modification to "get it right" – but these solutions still require technical software programmers to finalise and maintain the models.

So where are the toolsets that provide end to end modelling for use by subject matter experts rather than technical marvels used by genius software programmers?

2.2 Project Management and Development Methodologies

The traditional and still often typical software development lifecycle for large scale applications has long followed variants of the phases of; Specification or Requirements Analysis, Design, Implementation, Testing and Deployment with the project then entering a Maintenance phase – each Maintenance phase may also consist of similar lifecycles [35].

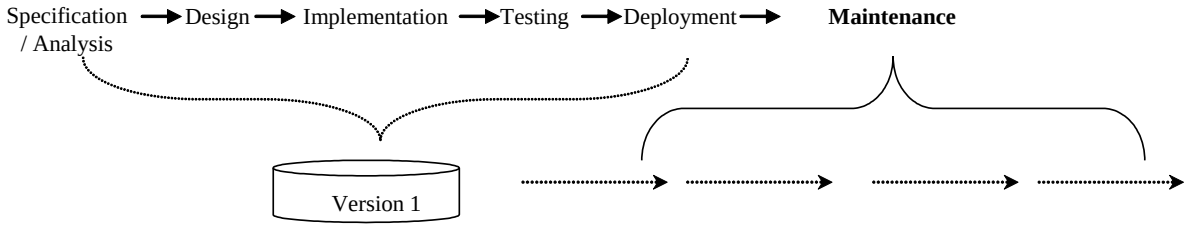


Figure 2 – Standard Development Methodology

There is a huge variety of project methodologies ranging from the generic PRINCE2 [37] and PMBOK [38] that are applicable to any form of project, to those specifically aimed at software development. Some of the more popular software development methodologies are:

2.2.1 Waterfall

The Waterfall model was originally derived from manufacturing processes by Bennington in 1956 [12] and is based on a linear or sequential series of phases similar to Figure 2– Standard Development Methodology . Work in a phase must be complete before proceeding to the next phase. It was later modified by Royce in 1970 to include feedback loops to provide for a level of revision and review [39].

[40] considers that the Waterfall model remains the most efficient way for creating software that provides back-end functionality i.e. with stable functionality that would be expected to remain static for long periods, such as relational databases, compilers or secure operating systems.

Accordingly the methodology is specification driven and is well suited to a fixed scope [13], rather than higher risk projects where the requirements are relatively unknown or can be expected to be subject to a high level of change.

2.2.2 B-Model

The Waterfall model was extended in 1988 by Birrel and Ould [14]. The B-Model extension was to include the operational lifecycle as a continuation of development lifecycles.

The extension was used to ensure that the constant improvement of software would be considered a part of the ongoing development process, providing evolutionary enhancements.

2.2.3 Spiral

The Waterfall model was modified by Boehm in 1986 by introducing an iterative approach that progressively spiral out encompassing the development of additional features.

The Spiral model combines the sequential discipline of the Waterfall model with an incremental or prototyping approach and is often used for high risk projects. It is based on successive loops through four phases or quadrants; Objectives, Risk Analysis, Development, and Planning [15].

At the completion of each cycle (or spiral), a new prototype is created for review and verification. Risk management is continually used to control the effort and scope of each spiral.

The additional benefits provided by the risk driven Spiral model include a focus on containing project costs and risks. Difficulties with Spiral include the requirement for adaptive project management, flexibility with stakeholder engagement, and the application of appropriate risk management [40].

2.2.4 V-Model

Developed by NASA in 1991 [16] as a variation of the Waterfall model, the V-model attempts to overcome perceived limitations of the Waterfall model by ensuring testing (and therefore problem detection) occurs earlier in the lifecycle.

Focusing on various activities like preparation of the Testing Strategy, Test Planning, creation of test cases and scripts in parallel to the development activities [41], a key strength of the V-model is its use in larger projects with multiple disparate stakeholder involvement.

The left leg of the V shape encompasses the requirements evolution, the apex the development, and the right leg the subsequent integration and verification steps. A vertical axis represents the level of composition with feedback loops across the V from the verification.

Modifications known as the V+ model extended the recommended usage to the development of front-end style applications by adding more user involvement. Additional modifications known as the V++ model added decomposition processes for analysis and resolution to the left leg, plus verification analysis to the right leg, to increase the scope of applicable projects to include service and business logic oriented style applications [42].

2.2.5 Unified Process Model

The (Rational) Unified Process (RUP) model is use case driven and iterative – it was developed in the 1990's by Rational Software to address the development requirements of object-oriented software [43].

The process uses models defined using the Unified Modelling Language [17], a collection of semi-formal graphical notations [44] such as use-case, activity, class, object, interaction and state diagrams providing a functional view obtained by modelling expected user interactions (see 2.5.1 OMG, MDA and UML).

RUP is iterative consisting of successive passes through the requirements, analysis, design, implementation and test phases whilst constructing appropriate UML

models. The process consists of 4 phases; inception, elaboration, construction and transition.

[40] considers that RUP is more suitable for business logic and front-end applications.

2.2.6 Agile

Agile methodologies are derived from a philosophy to promote "individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan" [45].

Agile refers to a multitude of software development methods based on iterative and incremental development, where requirements and solutions evolve through collaboration between cross-functional teams. It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change [18]. Common Agile methods include eXtreme Programming, Scrum and Lean Programming.

Agile is recommended when exacting requirements are unknown or difficult to elicit - Agile methods can "help to succeed in unpredictable environments" [46], they "concentrate on significantly improving communications and interactions among all stakeholders, promote constant feedback" while acknowledging that "agile methods are not a silver bullet and agile practices only work in context" – indeed it is establishing this appropriate context that is of utmost importance. [47] is concerned that agile methods "focus narrowly on the software to be developed and do not take a systems or engineering view of the development" resulting in "many software failures are caused by limiting the consideration of system stakeholders to the software developer and the customer".

2.2.7 Scrum

Is a form of Agile development based on defined "sprints" with the aim for the assigned team to create a shippable product increment, from the product "backlog", the prioritised requirements.

A key principle is its recognition that during a project the customers can change their minds about what they want and need. It adopts an empirical approach - accepting that the problem cannot be fully understood or defined, focusing instead on

maximizing the team's ability to deliver quickly and respond to emerging requirements [48].

Scrum (and other Agile methods) relies heavily on ensuring that a high level of customer involvement is maintained. The "sweet spot" for Scrum style projects involves "a small, co-located team; an on-site or available customer representative; an emphasis on coding and testing early; and frequent feedback into updated requirements" stressing the preference towards smaller projects and teams, a la flexibility. [49] provides guidance on how to try to upscale agility to larger projects.

2.2.8 Rapid Application Development

Can be defined as "any software life-cycle designed to give faster development and better results and to take maximum advantage of recent advances in development software"[23].

Primarily introduced by Martin in 1991 [50], RAD focuses on prototyping and iterative development in a collaborative environment with active participation of business stakeholders. In addition to the core RAD technological components that are expected to reduce the overall development there is an inherent assumption behind RAD that it is also an "iterative" and "prototyping" process [51] – relying on rapidly producing demonstrable functionality until a desirable and working system has been created.

RAD utilises 4 phases; requirements planning, design, development, testing and cutover. It has since evolved as a term to encompass many methods and methodologies that generally seek to speed application development through a combination of methodologies and software frameworks.

As RAD can often refer to any type of coding or development accelerant it may include; Computer Aided Software Engineering tools, reverse engineering, prototyping and Agile methodologies, code and components and re-use. Many IDEs include options for additional RAD components.

The basis of RAD was to counter the major problems of Waterfall style methodologies - that large applications took so long to build that the requirements may have changed enough to render the solution inadequate or unusable. [52] cites growing evidence that RAD style implementations can provide significant improvements in the speed of software development although conversely [53]

cautions that developers need to concentrate on “value engineering” to ensure that superfluous features are not included unnecessarily.

2.2.9 Extreme Programming

Promotes high customer involvement, rapid feedback loops, continuous testing, continuous planning, and close teamwork to deliver working software at very frequent intervals. It is based on 12 supporting practices of; planning, small releases, customer acceptance tests, simple design, pair programming, test-driven development, refactoring, continuous integration, collective code ownership, coding standards, metaphor, sustainable pace. The customer works very closely with the development team to define and prioritize granular units of functionality referred to as user stories [19].

However, [54] notes that he use of pair programming can lead to large increases in personal costs, while test-driven development also adds to the development effort. The main claim of XP is that this increased cost is more than compensated by three factors:

- A pair of programmers has a higher development speed than a single programmer.
- Continuously checking the code against the test cases improves the quality of the code.
- The code produced by a pair of programmers has a reduced defect density.

Do the claims stack up in reality? [55] reported a study where programmer pairs completed tasks 29% faster than single programmers, while [56] reported a range of 20-40% faster along with 15% fewer defects.

[57] reported simulation findings that diligent application of all XP practices could achieve a circa 28% reduction in overall effort, certainly a solid improvement. However, they also noted that XP also worked best in niche conditions: “should consider applying XP if the market pressure is strong, his programmers are much faster when working in pairs as compared to working alone, and there is a sufficiently large workforce available to run the project with the maximum number of pairs”.

2.2.10 Dynamic Systems Development Method

Is an agile delivery framework built on 8 working principles supporting five phases; Feasibility, Foundations, Exploration, Engineering and Deployment. It is

recommended for creating solutions that are required quickly. It has been formally documented and is freely available for use by the DSDM Consortium [20].

As a clear sign of industry acceptance DSDM Consortium, APMG International, the licence holders to the PRINCE2 project management methodology, jointly offer Agile Project Management certification in DSDM Atern [21].

A key quote from the DSDM Consortium "At DSDM we have always recognised the need for effective Project Management to provide governance to iterative development projects, including those applying Agile practices. The Agile Project Management Certification is a major step forward for both the Project Management and the Agile Communities as it provides the means to deliver Agile Projects in organisations requiring standards, rigour and visibility around projects, while at the same time enabling the fast pace, change and empowerment provided by Agile" [58].

The development of newer object based technologies and their growing adoption, particularly in the nineties has led to new system development methodologies such as Prototyping, Agile Processes, Big Ball of Mud [59], [60], [61], [53]. These methodologies take advantage of specific technology features as well as typically proposing differing levels of task decompositions, parallelism and customer interaction in order to target development efficiencies.

The newer generation of methods can provide specific advantages when dutifully employed [62] but they are not guaranteed to change the magnitude of the total system development effort [52] – much of the opportunity for optimisation is still reliant on strong management practices and technician performance, and the use of new technology - issues which are a fairly common pre-requisite for success in any project.

To an extent each of these methodologies are fundamentally waterfall in nature, although each with a differing level of focus. Whether they are iterative or incremental they still tend to necessarily follow the same general set of phases in determining requirements, planning and priorities, building, testing and ultimately deployment – some work in smaller blocks and more frequent cycles but at the end of any software development project run by different software development methodologies the output would largely be expected to be similar in terms of functionality as a similar solution to a problem would likely be developed.

While variations exist in different methodologies in the effort, time and applicable quality in developing a solution there is no obvious methodology that can consistently offer significant development savings. Of course there will always be variations introduced by more experienced teams and project managers and some methodologies will better suit some teams and organisations than others but overall the average or expected variations would tend to be within a common range over time.

The issue of traditional vs Agile methodologies does not yet have any definitive winner in terms of a single best fit. Unfortunately the debate is too often inflamed by their proponents by using "extreme and biased terms" and justifications "through either experience-based explanation or inadequate comparisons between the methodologies" provoking [63] to develop an objective framework to map the relationships between traditional and Agile methodologies.

While the above issue is unresolved, what is clear is that the various methodologies have specific benefits (and weaknesses) in specific project contexts however none can lay claim to providing clear orders or factors of improvement in terms of overall effort and time.

[64] is concerned about how well methodologies have been implemented and effectively used and sees that we are now in a post-methodology era driven by developer backlash against formal methodologies - where there is no clear pathway and diversity is the driver.

Is this good enough? Clearly any incremental development savings are worthwhile for an organisation and when adopted by many organisations the overall benefits are increased but overall it is still only incremental improvements for which I feel that better opportunities may exist.

So while project management and development methodologies can aid in at least trying to successfully complete a software development project, it is our belief that other fundamental changes have to occur in how large scale applications are developed in order to achieve major efficiencies and reductions in development effort.

2.3 Software and Technology Advances

Prior to the emergence of the Internet there were considerably lesser common development technologies although the more closed environments propagated a greater variance as vendors were more able to maintain a higher level of dependence.

Technology advances have evolved particularly dramatically since the emergence and universal accessibility of the Internet.

In many ways the Internet led advances have been sideways, as new methods of providing application content over initially slow bandwidths required drastic re-engineering efforts. It became commonplace that organisations would progressively require web sites for promotion and commerce, also enabling entirely new e-commerce only industries.

New technologies and programming languages were developed to support the progressive de-centralisation and distribution of systems and users. With increasing bandwidth availability so have the toolsets evolved to provide richer access functionality in terms of multimedia as well as more feature rich application features.

Other sideways evolutions that have occurred involve the proliferation of new classes of remote and portable computing devices such as smartphones and tablets. In particular the popularity of these devices have also promoted significant global duplication in that web sites were no longer enough – device specific applications or "apps" became the new additional requirement, although with a variety of major competitors in the app space, an app for each major device type would also be required to maintain an adequate Internet presence. Many new technologies have been developed to address the application and information requirements of this new connected Internet age.

However, in many ways, it is not so much the change from the new technologies themselves but how the technologies and the new open environments offer better and more efficient ways to enable software development in a more collaborative way supported by often immediate communications and data transfer.

2.3.1 Software Reuse

Software reuse is a general concept of developing software that can be re-used for other purposes, either directly or with modification. Software reuse is of supreme importance because it has the potential to yield enormous economies of scale.

[65] defines three ways of defining software reuse:

- **Static Reuse**: can be defined in terms of the number of source code references to my component, or the number of software items that refer to my component. Static reuse generates application benefit, in terms of faster development and/or easier maintenance

- **Deployment Reuse**: can be defined in terms of the number of consumers with access to services provided directly or indirectly by my component.
- **Dynamic Reuse**: can be defined in terms of the frequency of execution of my component.

Business benefit comes from high levels of deployment reuse and dynamic reuse. This can often be achieved without high levels of static reuse. However, a lot of software engineering is focused on static reuse [65].

[66] recommended method is through software reuse. The primary efficiency consideration of software reuse was initially fuelled as organisations' computing based information systems infrastructure became internally interconnected with corporate networks and newer object based development technologies were adopted, [67].

In [68] they show that a pragmatic software reuse can:

- significantly decrease the time that developers require to perform pragmatic reuse tasks,
- increase the likelihood that developers will successfully complete pragmatic reuse tasks,
- decrease the time required by developers to identify infeasible reuse tasks, and
- improve developers' sense of their ability to manage the risk in such tasks.

[69] recommends on how to construct global software reuse repositories as not all software will always be relevant to all users due to inherent language and interface incompatibilities plus any associated redevelopment concerns.

Clearly, encouraging as much software reuse as practicable will always reap ongoing benefits as long as the fundamental economic benefit is maintained by each use case: the cost of procurement plus integration is less than the cost of any equivalent new development. When multiplied by the number of end users the global benefit can be significant.

Commercial options or extensions to the IDE or development environment are often available that provide pre-coded functionality as coding accelerants – these may be provided by the original vendor or by specialist third party library developers that aim to provide substantial additional capabilities via their pre-developed extended functionality libraries providing faster or higher quality development.

The Internet age has also given rise to an even greater availability of coding shortcuts. Many organisations and individuals publicly make available their software source code and libraries for general use, often royalty free. Regardless of whether the individual open source providers have based their reasons on political, social, ego, marketing or financial motivations, there is a massive availability of free or minimal cost software available from the open source community ranging up to fully commercial quality high feature components.

The sensible use of pre-built or readily modifiable software components has the potential to both increase the functionality and reduce the development effort of software projects.

2.3.2 Component Based Development

A software component can be any software package, web service, web resource or module that encapsulates a set of related functions or data, with a focus on the generic connectors that compose the components into one unit [70]. Reusability is an important characteristic of a high-quality software component. Programmers need to design and implement software components in such a way that many different programs can reuse them.

[71] advocate component based development - that is to construct computer based information systems typically involves a number of potentially independent components. The physical components of an information system such as the user interface, business logic processing system and database server can utilise completely different development technologies and deployment platforms Development methodologies provide guidance for the overall projects but do not typically dictate the choice of technologies and platforms.

Utilising component based software offers several advantages and encourages the move towards more modular systems built from reusable software artefacts - it is expected to enhance the adaptability, scalability, and maintainability of the resultant software [72].

Constructing systems with pretested software components is also likely to improve software quality and reliability and has the potential to increase developer productivity[73], shorten development life cycle, reduce development costs and generally move software development from a craft to a more robust industrial process [74].

[72] noted the following challenges for component based development to be effectively utilized:

- methods and procedures for determining component requirements have to be developed.
- effective methods to search component repositories are required.
- component markets that allow easy procurement of components and frameworks must evolve.

As a structured yet general example of software reuse the application of component based software usage, where the required components are readily available with a compatible technology and appropriate functionality, is virtually an essential requirement to minimise the costs of software development.

2.3.3 Middleware

[75] describes functions of middleware as being able to "adapt their structure and behaviour at runtime" or "identify changes in the environment that can affect the application, in order to perform adaptations in a transparent way", or by [76] as "an intermediate layer to abstract the homogeneity and hide the difference of underlying systems, can be used to reduce the complexity for Internet application development.".

When utilising smaller or lower complexity compatible components in a software development environment the "middleware component" used to integrate the core application functionality might be as simple as few lines of code for each instance of a utilised component.

However, when utilising more architecture-level components with complex functionality the use of more structured middleware technologies such as CORBA (Common Object Request Broker Architecture) [77], COM (Component Object Model) [78] and RMI (Remote Method Invocation) [79] becomes an essential toolset to provide services for enabling component composition and interactions [80].

Ensuring interoperability is a critical issue with utilising heterogeneous software components. The variety of middleware vendors led to incompatible, proprietary component and middleware standards - components "speaking" the same language are interoperable, while those "speaking" different languages are not [80], requiring the use of separate integration solutions or specialised universal middleware bridges or service-oriented middleware solutions that provide mechanisms for service coordination and cooperation - these composites can potentially act as interoperability

bridges between services running on heterogeneous conventional middleware platforms [81].

Just as software in different languages can be incompatible, so does interoperability remain a fundamental problem for distributed systems due to the increasing level of heterogeneity and dynamism of the networking environment - [82] offers emergent middleware that is synthesized on the fly according to the behaviour of the associated networked systems, using ontologies in the middleware design so that middleware may dynamically emerge based on semantic knowledge about the environment.

2.3.4 Frameworks

[83] defines a software framework as providing "an API that encapsulates and hides the concept of service from the remainder of the application, supports service selection and execution driven by application business rules and reduces development effort".

[84] Hamu & Fayad, Madsen et al's method is based on the creation of frameworks that allow extension and reuse. The progressive management of component software modules that could be reused and extended between projects helped foster the creation of frameworks – fully featured reusable and extendible development environments that are partially "pre-built" from flexible components that can speed development considerably [85] [86].

Frameworks thus provide examples of multiple component sets that offer widespread functionality across a broad domain or specific feature set. Many software development frameworks are available for the common technologies e.g. web application frameworks include support for: PHP (PHP: Hypertext Preprocessor), Java, .NET, Ruby, Python, JavaScript, CSS (Cascading Style Sheets). Frameworks can often be specific to a particular technology or problem domain often requiring developers to master multiple frameworks. Additionally, frameworks do not often provide a full feature set to satisfy efficient development, requiring additional component libraries.

Frameworks represent a solid working environment to directly support the software coding and development of an application and can optimise much of the arduous hand-coding issues that are required. A framework's support of application modelling is in terms of the nature, structure and syntax of their underlying supported

programming languages, typically representing useful accelerants in producing the required code for a defined problem.

2.3.5 Application Layers and Replacements

Examples of major homogeneous application of components is in the large scale or wholesale replacement of application layers in software development. The common allocation layers are; the presentation, business, and data layers [87]. This is often referred to as a 3-tier software architecture.

[88] expands upon these basic 3 tiers to encompass the concepts of N-tiers, where "every tier is populated with components and every component may include parts of the application logic".

Commonly used technologies such as SQL (Structured Query Language) and RDBMS (Relational Database Management System) solutions seek to abstract away the concerns of managing physical data. Similarly, web and XAML (Extensible Application Mark-up Language) style technologies can be used to manage the user interface artefacts in an application rather than individually creating specific form style objects. Options exist to supplement the business logic layer but as this layer is the most application-specific these instances tend to be the exception rather than the rule.

The use of application layer replacements can and has provided many development accelerants by reducing efforts in continually replicating similar code for the management of different objects (but of a similar nature).

While an application layer replacement may reduce significant effort, it still requires an appropriate management and control interface between the core application and each layer technology. These interfaces can additionally require significant effort to initially construct and maintain.

2.3.5.1 User Interface Layers

The Model-View-Controller (MVC) software architecture separates the representation of information from the user's interaction with it, such as "capturing all concerns in a model or in its dependencies through highly-decorated classes and fields; e.g., domain logic, database access, or field presentation widgets" [89]:

- Model consists of application data, business rules, logic, and functions.

- View can be any output representation of data, such as a chart or a diagram.
- Controller mediates input, converting it to commands for the model or view.

The most widespread application of MVC is observed in everyday usage of the Internet. The latest version, HTML5 (HyperText Markup Language 5), is the standard mark-up language for structuring and presenting World Wide Web (WWW) content [90]. HTML5 empowers browsers to become a suitable platform for developing rich Web applications [91].

Microsoft's XAML [92] is targeted towards Windows based systems only and the XAML specification does provide for a rich UI (User Interface) feature set. XAML relies on explicit object bindings to application logic objects and thus does not seek to achieve nor provide platform independence. Alternative open source and commercial releases of XAML-based products include MyXAML [93] and Xamlon [94].

The creation of XUL (XML User Interface Language) [95] by the open source community is supported on a wider range of platforms than XAML – extending from Windows to include Apple Macintosh and various popular strains of UNIX and Linux operating systems. The platform specific runtime support for XUL is provided as part of the Mozilla project Firefox browser [96]. XUL also utilises direct object bindings to local application objects and is thus platform dependent and does not directly support separation of the UI layer from the application.

2.3.5.2 Business Logic Layers

[97] defines business logic as "the programming that manages communication between an end user interface and a database. The main components of business logic are business rules and workflows. A business rule describes a specific procedure; a workflow consists of the tasks, procedural steps, required input and output information, and tools needed for each step of that procedure. Business logic describes the sequence of operations associated with data in a database to carry out the business rule."

The specialist application logic required for most software will be defined in and constitute the business logic layer. For the majority of applications this will often be a custom code-base implemented specifically to solve the core business problems, optionally using third-party components as development accelerants.

Accordingly there exists a wide variety of different implementations of business logic layers:

- **Client / Server**: no acknowledged business layer may exist where all application logic resides in the client application and/or database.
- **Consolidated**: where all of the business logic resides in the business layer (although considered the ideal, often some proportion may be duplicated in other layers for performance reasons).
- **N-Tier**: and somewhere in between where the business logic is apportioned between all of the application layers.
- **Enterprise Business Rules engines**: can be utilised to partially or fully implement the modelled business logic in an easily managed and flexible repository (although usually requiring additional interface constructions). These may be modelled using workflow based composition languages such as Business Process Execution Language (BPEL) [98] or BPML (Business Process Modelling Language) [99].
- **Custom Code**: is the common coding glue to bind and interface components and construct additional business logic as required.

There is an enormous range of available rules engines and component libraries that can be used to construct and maintain an enterprise's applications business layers, with an equal diversity amongst developers.

2.3.5.3 Data Layers

The definition and implementation of SQL as the de facto standard for database storage, access and manipulation was the first major separation of processing responsibilities [100]. Managing database transactions using SQL provided both portability, as more database management systems adopted SQL, and ultimately transparency as the use of vendor independent SQL code allowed generic access to varied and distributed SQL based RDBMS.

Further extensions of the basic RDBMS to more closely support object-oriented modelling techniques were refined in Object-Relational DBMS (ORDBMS) and Object-Oriented DBMS (OODBMS). The ORDBMS supports modelling basic object behaviour such as; complex data, type inheritance, and object behaviour [101]. The OODBMS extends the management of objects within the repository to further allow

object modification and creation and maintain consistency between the developers object oriented programming environment [102].

More popular extensions to resolving object relational database transaction automation have resulted in the development of Object Relational Mapper (ORM) technology, such as the open source product Hibernate [103], that assists in removing or reducing the platform dependence issues of alternate database server implementations, enhancing automated transaction processing capability and further simplifying database access and compatibility from within applications.

The major impediments are schema dynamism due to application or database evolution, dynamism is a common problem to all fixed interfaces. [104] proposes self-configuring ORM components, which reflectively configure the persistency layer usage sites, thus leading to improved maintainability of software - a self-configuring component analyses the actual persistency layer usage pattern. Based on this information, the actual queries are configured.

[105] proposes how fully distributed architectures and applications should demonstrate a ”liquid” architecture where “functionality and data are completely freed from any fixed locations or functional paths and may flow at will”. Such an environment would potentially require minimal programming due to a comprehensive availability of pre-existing integrated components. It should also rely heavily on the use of models to populate the required logic and interactions of the components.

The emergence of many global standards for data access and interchange as part of the evolution of the Internet has facilitated previously unimagined interoperability and information exchange capabilities. While these capabilities have provided entire new e-commerce industries and supported consumers and employees with undreamt of information access they have also attracted a major global cost with organisations developing web site and e-commerce presences as well as often multiple platform versions of apps, and the modification of re-engineering of enterprise information system applications to operate in a remote and distributed environments.

The widespread growth of components, frameworks and application layer component replacements has supported the multitude of new technologies and platforms required to maintain this global infrastructure, and necessarily provided the

development accelerants required to establish and maintain this surge of new capability and software developments.

However advantageous the features provided by the new generations of application access the bottleneck remains that technical programmers are still required to plug-in, integrate and customise the great bulk of the software that is not readily componentised.

Clearly, the widespread availability of readily accessible and functionally useful reusable components is a key requirement to continue to reduce ongoing software effort but we need better ways of gluing them together so that they can be more universally accessible.

I believe that minimising these efforts can be achieved with significant benefits by a greater use of modelling rather than programming, as the programming is so often a massive duplication of effort in terms of; CRUD, integration, organisational replication, and even vendor replication.

2.4 Systems Development Processes and Tools

How do we best organise our work to be done for development?

IDE's are great for the functions they provide – but code IDE's are granular to the syntax of the code and help greatly with arranging structure it is often the supporting syntactic structure for the codebase rather than relating to the underlying problem or application function that is actually required – this all has to be worked through by those technical specialists that understand the IDE and programming language specifications – and they aren't usually those well versed with the nature of the business problems.

Technical and business process standards have been developed to aid both the technical development of the software and to aid in mapping out the business problems to help with educating the technical programmers with the nature of the problem to be converted into program code – attempts to translate from business concepts and language to computing features and programs – like all translations, information can and is often "lost in translation" but at least any process that attempts to capture and document requirements in any model, ontology or increasing granularity is a positive – BPEL etc.

CASE tools can provide substantial benefits for the subject domain that they service. Essentially an IDE for their target model, CASE tools represent a next evolution to capture and model domain requirements. Commencing initially with data modelling, then progressing to object modelling and business processes. CASE tools can provide great accelerants for modelling, managing and generating their domain environments.

Launched by the OMG in 2001, Model Driven Architecture seeks to further extend the capture of application requirements into models that can be subsequently transformed into executable code. MDA specifies guidelines for the structuring of models, primarily using UML, to capture a Platform Independent Model (PIM) for later transformation into a Platform Specific Model (PSM) with vendor support to then generate various forms of output code or templates.

2.4.1 Integrated Development Environment

The most fundamental software coding is where all software is developed using the basic Integrated Development Environment (IDE) as provided by a chosen vendor of the core software development tools. Such an environment will typically only provide the editor, compiler and debugger to allow software programs to be created. Using only an IDE programmers must develop all code from scratch. IDE's may also provide a set of pre-built components or allow for additional sets of components to be integrated thus becoming more of a framework.

IDE's can be extremely simplistic such as Microsoft's Small BASIC (Beginner's All-purpose Symbolic Instruction Code) [106] extending up to the multi-team collaboration environments provided by the extended versions of Oracle's Java Enterprise Edition [107] and Microsoft's Visual Studio [108] and Team Foundation Server [109]. Many IDE's are also supported by extensive third party component libraries to extend the feature set provided by the OEM (Original Equipment Manufacturer).

IDE's generally provide an adequate development platform for the features that the IDE provides, for use by the trained (or learning) technical programmer. Code generating IDE's are by nature granular to the syntax and structure of the supported code base and language specification, however, these supporting syntactic structures relate to the codebase requirements rather than necessarily relating to the underlying problem or application function that is actually required.

Translating and resolving business problems has to be worked through by those technical specialists that understand the IDE and programming language specifications – typically these roles are distinct and separate from those that are well versed with the nature of the business problems. Thus are required the methodological processes and supporting roles such as requirements analysis and design, and for business and systems analysts to assist with capturing and translating requirements into formats that the more technically focussed specialists are more familiar with, to then proceed with coding and developing the required application software using the IDE toolsets.

2.4.2 Business Software and Process Related Standards

[110] defines a standard as "something considered by an authority or by general consent as a basis of comparison; an approved model". In general standards are needed for consistency to ensure mutual understanding – in computing adherence to some standards can become an essential requirement due to the often precise syntactic requirements of the underlying computing environments.

The most fundamental standards relate to the binary logic states of the computer hardware however the majority of software developers need only concern themselves with the higher level model abstractions that they work with on a regular basis – the syntax and structures of the programming languages and supporting IDE's.

As programming languages typically provide low level general purpose operations, higher level abstraction models have been required to model more conceptual, process and naturally human oriented activities. This higher level modelling can thus be performed by Subject Matter Experts (SME) from the relevant domain who are fluent in the underlying real practices.

In computing and related fields, many technical and business process standards have been developed to aid both the technical development of the software and to aid in mapping out the business problems using domain specific models. Additional manual or automated transformations can then be performed from the domain specific models into more technically focussed models, thus assisting technical programmers with the non-technical nature of the problem to ultimately be converted into program code.

Such transformation attempts to translate from business concepts and language to computing features and programs, is not always perfect nor are all possibilities always

supported and as in all translations, information can and is often “lost in translation”. However, at least any process that attempts to capture and document requirements in any model, ontology or increasing form of granularity is a clear positive and will promote successive evolutionary improvement to clarifying requirements and seeking their ultimate software instantiation.

Some of the more accepted standards in use for capturing requirements and developing software applications are:

Business Process Model and Notation (BPMN): is a graphical representation for specifying business processes maintained by the OMG [111]. BPMN 2.0 released in 2011 has extensive third party toolset support however it also requires other modelling notations in order to fully model all types of processes [112].

Business Process Execution Language: established by the Organization for the Advancement of Structured Information Standards (OASIS) [113] as an executable language for specifying action within business processes with web services [114]. BPEL is a language specification with no standard graphical notation although third parties have introduced their own partial versions. [115] have uncovered several ambiguities and inconsistencies in the data models and type systems exposing mapping flaws.

Web Services Choreography Description Language (WS-CDL): is a specification by the World Wide Web Consortium (W3C) defining a XML-based business process modelling language.

XML Process Definition Language (XPDL): has been standardised by the Workflow Management Coalition (WfMC) [116] to interchange business process definitions between different workflow products [117]. The XPDL technology introduces tools that allow for a direct mapping of complex processes to software logic [118].

Architecture of Integrated Information Systems (ARIS): is a “multiuser platform for definition and analysis of the workflows in the business organization, supporting development of complicated heterogeneous IS (Architecture of integrated Information Systems) and escorts the complete cycle of development” [119]. Limited third party toolset support is available to transform the models to BPMN and other formats for development. [120] proposes a SOA based ARIS model for business process re-engineering to overcome the implementation support for ARIS.

Java Process Definition Language (jBPM): is an open-source workflow engine written in Java that can execute business processes described in BPMN [121]. Amongst other platforms it is supported by the popular open source JBoss Java based application server.

Unified Modelling Language: now managed by the OMG, UML is a general-purpose modelling language for software engineering - it provides "graphical approaches to requirements elicitation" [122] with a set of graphic notation techniques to create visual models of object-oriented software. UML has a wide range of toolset support from third party vendors, and is considered by many as a de facto standard software design language [123]. However, common and major criticisms of UML include; lack of formal semantics, expressiveness, customisability and completeness and consistency [124].

Some of these standards and their associated supported toolsets are aimed squarely at technical system designers and developers while others provide a higher level business process focus with tools that are accessible to non-technical business analysts. [125] concludes that "much of the effort in creating software development methods has been focused inwardly towards the needs of the software development teams, with less consideration for the needs of the non-specialists. Ultimately, the effectiveness of any development approach will be judged by those who need to use it".

I agree with the above summation - traditionally the majority of the software design tools have been targeted towards the technical specialists that are tasked with developing and coding the application software. While more business-process centric effort has since been expended on capturing business requirements and processes their outputs tend to be inputs for the system developers rather than direct generators of the applications themselves – a focus I believe is needed to be far more efficient overall.

2.4.3 Computer Aided Software Engineering

Can be defined as "the use of a computer-assisted method to organize and control the development of software, especially on large, complex projects involving many software components and people. Using CASE allows designers, code writers, testers, planners, and managers to share a common view of where a project stands at each stage of development" [24].

Major types of CASE tools include; business process engineering, configuration management, database management, documentation, interface design, process modelling, programming, requirements tracing, software analysis and design, test management and web development.

By capturing and managing the CASE tool's knowledge domain in a structured format the CASE tool is also available as a multi-user collaboration environment. Most of the previously mentioned major standards are supported by various CASE tools to aid in capturing their requirements specifications and ensuring the appropriate structuring of precise output formats for generation or as input to other CASE tools for further refinement.

CASE tools tend to specialise on the specification of a particular layer in the software development process and as such can require a great deal of training and education in order to be able to be used effectively by their related domain experts.

[126] considers that the domain of CASE tools were those that arose during the 1980's and 1990's (as a precursor to 2.4.4 Model Driven Engineering (MDE)) and cites numerous overall shortcomings affecting their adoption and usage:

- Inadequate modelling for features such as; distribution, fault tolerance and security required significant additional coding,
- Inability to scale to complex, production-scale systems in many domains,
- Typically not supporting concurrent engineering, too often limited to single user access.
- Targeting proprietary execution environments making it hard to integrate the generated code with other language and platforms.
- Lack of support for many domains due to "one size fits all" graphical representations.

There are literally hundreds if not thousands of CASE tools that have been developed commercially as well as academic research prototypes. Some of the major CASE / modelling tools in commercial use are listed (alphabetically) in Table 1 - List of Popular MDE Tools .

2.4.4 Model Driven Engineering (MDE)

Model Driven Engineering can be defined as "a software development methodology which focuses on creating and exploiting domain models (that is,

abstract representations of the knowledge and activities that govern a particular application domain), rather than on the computing (or algorithmic) concepts" [32].

MDE toolsets and development environments first capture the application design requirements into a model and then generate the appropriate source code and software artefacts for creation of the final application executable. Many RAD and CASE tools provide examples of these toolsets. Within the feature scope and presentation layouts supported by the toolsets, specific application subsets can be efficiently developed with these tools for a variety of local execution and remote Internet environments.

[126] notes that MDE technologies need to combine:

- **Domain-Specific Modelling Languages (DSML):** whose type systems formalize the application structure, behaviour, and requirements within particular domains. DSMLs are described using meta-models, which define the relationships among concepts in a domain and precisely specify the key semantics and constraints associated with these domain concepts. Developers use DSMLs to build applications using elements of the type system captured by meta-models and express design intent declaratively rather than imperatively.
- **Transformation engines and generators**: that analyse certain aspects of models and then synthesize various types of artefacts, such as source code, simulation inputs, XML deployment descriptions, or alternative model representations. The ability to synthesize artefacts from models helps ensure the consistency between application implementations and analysis information associated with functional and QoS (Quality of Service) requirements captured by models.

A vexing problem in RAD, CASE and MDE tools and modelling and generation in general is that the functionality of the generated applications is usually limited by the scope of the available functionality supported by the toolset.

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 [127] who specifies alternate aspects and [128] who identifies model insertion points for code insertion. Typical bidirectional solutions to allowing modifications to the generated output involve allowing direct modification of the generated source code, or allowing

for embedding external objects or links to external code to provide the required features that cannot be provided by the modelling toolset.

[129] base their works on the UML 2 specification to seek to reduce coding and transform models of business processes into executable forms. [130] takes a strong model generation approach that seeks to identify customizations to a base model but then implements and maintains each new customized model as separate models executed as individual application instances.

In [131] 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. Such a model is the goal of our temporal meta-data framework for EIS applications.

Some of the major MDE tools in commercial use are listed (alphabetically) in Table 1 - List of Popular MDE Tools . The table indicates the broad range of functions that are provided by each MDE tool:

- **Requirements**: if Requirement Traceability features are supported.
- **Process**: if visual process mapping in any formats are provided.
- **UML**: if any UML models are supported.
- **Schema**: if database schemas, entity / relationship modelling is supported.
- **Code Gen**: if the tool generates source code, scripts etc either in full, in outline or via pattern templates.
- **DSM**: if the tool supports DSM (Domain Specific Modelling) to define and model additional domains.

Tool	Vendor	Req'ments	Process	UML	Schema	Code Gen	DSM	URL
AppComplete / ECO	CapableObjects	√	√	√	√	√		http://www.new.capableobjects.com/
Artisan Studio	Atego	√	√	√	√	√		http://www.atego.com/products/artisan-studio/
AtomWeaver / ABSE	Isomeris	√	√		√	√	√	http://www.atomweaver.com/
CA ERwin Data Modeler	CA Technologies				√			http://erwin.com/products/data-modeler
CaseComplete	Serlia Software Development Corp	√	√					http://www.casecomplete.com/
CoCoViLa	Tallinn University of Technology		√			√	√	http://www.cs.ioc.ee/cocovila
CodeFluent Entities	SoftFluent		√		√	√		http://www.softfluent.com/products/codefluent-entities
DB-MAIN	REVER				√			http://www.db-main.eu/?q=en
DeZign for Databases	Datanamic Solutions				√			http://www.datanamic.com/dezign/
Eclipse Modelling Framework	Eclipse Foundation		√			√	√	http://eclipse.org/modeling/emf/

Tool	Vendor	Req'ments	Process	UML	Schema	Code Gen	DSM	URL
Epsilon	University of York		√			√	√	http://www.eclipse.org/epsilon
ER/Studio	Embarcadero Technologies		√	√	√	√		http://www.embarcadero.com/products/er-studio
GenerateXY	DotXY		√			√	√	http://www.codeproject.com/Catalogs/3764/GenerateXY-Code-Generation-Studio.aspx
Generic Eclipse Modelling System	GEMS		√	√		√	√	http://www.eclipse.org/gmt/gems/
GeneXus						√		http://www.genexus.com/global/home?en
Graphical Modelling Project	GMP		√			√		http://projects.eclipse.org/projects/modeling.gmp
HyperSenses / ANGIE	DELTA Software Technology		√			√	√	http://www.d-s-t-g.com/en/hypersenses.html
Innovator	MID GmbH	√	√	√	√	√		http://www3.mid.de/en/welcome-to-mid.html
Iron Speed	Iron Speed Inc				√	√	√	http://www.ironspeed.com/products/Overview.aspx

Tool	Vendor	Req'ments	Process	UML	Schema	Code Gen	DSM	URL
LEONARDI Business First	W4	√	√		√	√		http://www.leonardi-free.com/ http://www.w4global.com/
MagicDraw / Cameo	No Magic Inc	√	√	√	√	√		http://www.nomagic.com/
MetaEdit+	MetaCase	√	√			√	√	http://www.metacase.com/products.html
Modeliosoft	Modeliosoft	√	√	√	√	√		http://www.modeliosoft.com/
MySQL Workbench	Oracle Corporation				√			http://www.mysql.com/products/workbench/
Navicat	PremiumSoft Cybertech Ltd				√			http://www.navicat.com/
objectiF	microTOOL	√	√	√	√	√	√	http://www.microtool.de/objectif/en/index.asp
Portofino	ManyDesigns S.r.l					√		http://www.manydesigns.com/en/home
Rational Rose / Rhapsody	IBM	√	√	√	√	√	√	http://www-03.ibm.com/software/products/us/en/ratirosefami/ http://www-03.ibm.com/software/products/us/en/ratirhapfami/
RISE Editor	R2B Software			√	√	√		http://www.risetobloome.com/Page_1_S_NoPadding.aspx?item=530

Tool	Vendor	Req'ments	Process	UML	Schema	Code Gen	DSM	URL
SAP Sybase PowerDesigner	Sybase Inc	√	√	√	√	√		http://www.sybase.com.au/products/modelingdevelopment/powerdesigner
SCADE Suite	Esterel Technologies	√	√		√	√		http://www.esterel-technologies.com/products/scade-suite/
Select Architect	Select Business Solutions	√	√	√	√	√		http://www.selectbs.com/analysis-and-design/select-architect
Simulink	MathWorks		√		√	√		http://www.mathworks.com.au/products/simulink/?s_cid=wiki_simulink_2
Sparx Enterprise Architect	Sparx Systems	√	√	√	√	√	√	http://www.sparxsystems.com.au/
StarUML	Open Source		√	√	√	√	√	http://staruml.sourceforge.net/en/index.php
TMS Data Modeler	TMS Software				√			http://www.tmssoftware.com/site/tmsdm.asp
Together	Borland		√	√	√	√	√	http://www.borland.com/products/together/
TOPCASED	TOPCASED	√		√		√		http://www.topcased.org/
Uniface	Compuware		√		√	√		http://www.compuware.com/en_us/application-development.html

Tool	Vendor	Req'ments	Process	UML	Schema	Code Gen	DSM	URL
Visual Paradigm for UML	Visual Paradigm International	√	√	√	√	√	√	http://www.visual-paradigm.com/product/vpuml/
YAKINDU Statechart Tools	Open Source		√			√		http://www.statecharts.org/

Table 1 - List of Popular MDE Tools

Each MDE tool provides either niche functionality for a particular aspect of the application development lifecycle, such as data schema only, or provides for multiple aspects, with some products close to covering the full lifecycle.

As a general rule, most products that seek to cover the full development lifecycle provide code generation as the ultimate output objective, as efficiency enablers for technical developers – much of the code generation capability may need to be further refined by each developer based on their own preferred design patterns. More of these products are now providing for round-trip management to more readily facilitate manual code modification of the output code.

Every product undoubtedly will provide some degree of efficiency gain via its modelling capability when used appropriately, whether targeted at specific roles or the majority of the development lifecycle effort. Some of the products have a more primary focus on using the model as the core of the ongoing application definition, with a greater provision on options for direct model execution environments which shares a common philosophy with the aims of this thesis. These products include:

- **LEONARDI**: consists of the Application Composer, as a model editor, with the Application Engine providing the runtime engine. While seemingly very functional, this is achieved through an interface and customisation capability strongly biased towards the Java code that is the fundamental execution output, thus requiring a higher degree of technical expertise.
- **Uniface**: provides an IDE as the editor and deployment tool, with separate runtime execution environments. Additional configuration and customisations are provided via Uniface's proprietary scripting language as well as via external plug-ins and web services. Uniface is reviewed favourably although still requires a higher level of technical knowledge to define the model.

Comments from case studies of these modelling tools include:

"Uniface is five times faster than Java"

"Uniface's productivity helps us to respond with agility to customer requests"

"a robust solution that can scale to support tens (even hundreds) of thousands of self-service users"

“We can now do releases every two to three months on just one installed instance, which means we can keep up-to-date with market requirements. Before … it was more likely to be one major release every few years.”

“Easy to adapt product to new requirements as architectural changes are minimal”

“Uniface gives you a standard way of building applications, so even if someone has written code differently from the way you would have done it yourself, you can understand it easily.”

“We sit down together to discuss the enhancement, and then I quickly prototype it in Uniface and show it to the users to see if it’s what they want”

“It would have taken us at least five years to take a Cloud product to market with .NET and C#. With Uniface, it took less than 18 months”

“cut costs by a third while paving the way for modernization and because it’s written in Uniface, minimal work has been needed”

“Our users benefit from the fact that we are able to make even complex changes to the application quickly using Uniface,”

“When programming the interfaces or creating the reports, we also use other development environments and programming languages, such as Microsoft .NET and Crystal Reports,”. When comparing these products with Uniface, notes that, “when working with other development environments, we see again and again that we are able to work more productively with Uniface.”

“Developer productivity doubled”

“With Uniface, we developed a product in two years’ time that surpasses what took 20 years to accomplish”

“maintain these applications with 16 employees. If I would do it with Java development, I guess I would have need for 40 or 50 employees.”

“Using the repository functionality of Uniface is key. We put all the necessary data definitions in a repository and don’t have to write source code. This way, when a change is needed, all we have to do is change the repository and the changes will be inherited by the application automatically.”

“Extremely reliable code base and graceful continuity from one release to the next”

"Upgrading from one version to the next is painless, which is something you rarely find with other products."

“finds that just two weeks’ Uniface training is sufficient to allow developers to become productive”

“Even though we have thousands of components, our use of Uniface means that we’re never faced with spaghetti code”

“we have a development environment that allows us to accommodate new requirements fast.”

“found J2EE development to be less productive because of its complexity. We could clearly see that the Uniface developers are two or three times more productive in terms of turning round solutions”

It is exciting that there are so many useful MDE style systems, especially those such as LEONARDI and Uniface that are maintaining some commercial success, while somewhat disappointing that there are so few model focussed development environments compared to the plethora of code generation tools. As the functionality offered by the tools increases, to minimise the need for code based customisations, and as computing hardware offers greater solution processing speed for direct model executions, perhaps this ratio will progressively reverse.

2.5 Collaborative Model Based Development Options

The MDE pathway represents a potential future that is strongly supported by this thesis. To move away from the mass coding and code duplication efforts, replaced by even better MDE style tools that can be used effectively by even non-technical users.

Modelling technology has always lagged behind computer system development, largely due to a lack of common standards, although many available standards are now in common usage (see 2.4.2 Business Software and Process Related Standards). As the technologies that are used and available have progressively merged or been provided with inter-communications and data-transfer capability so have the opportunities for modelling increased [132].

There are many discrete MDE tools available as partially listed in 2.4.4 Model Driven Engineering (MDE) . Additionally there are major industry modelling initiatives that seek to bring MDE opportunities into the mainstream such as the primarily commercially driven Object Management Group [133] with its Model

Driven Architecture [134], and the Eclipse Modelling Project (EPM) [135] which is largely supported by open source and academic groups.

2.5.1 OMG, MDA and UML

The aim of the Object Management Group is to "provide an open, vendor-neutral approach to the challenge of business and technology change". The OMG represent one of the largest proponent groups for MDE with the goal for their Model Driven Architecture initiative to "separate business and application logic from underlying platform technology". [134]

The OMG approach is predicated on the design of Platform Independent Models defined primarily with 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 also manages the standards, primarily:

- UML 2,
- MetaObject Facility (MOF) where models can be stored, shared and transformed,
- XML Metadata Interchange (XMI) for defining, interchanging, manipulating and integrating XML objects and data, and
- Common Warehouse Metamodel (CWM) as a standard interface to interchange metadata between warehouse tools, platforms and repositories.

A primary goal of the OMG is interoperability and the tools and technologies are primarily aimed at highly technical analysts and developers. The OMG supports industry developers of supporting toolsets as well as users developing with the technologies.

The OMG membership consists of well over 300 organisations and corporations that participate in development of their standards, guide the strategy direction and develop many optimisation and code generation tools aimed at the technical IT user.

Membership includes a veritable industrial who's who such as: Aberdeen Group, Adaptive, AT&T, Boeing, Business Architects Association, CA Technologies, CSC, Dell, Eclipse Foundation, Fujitsu, General Electric, Hewlett-Packard, Hitachi Ltd, Honda, Huawei Technologies Co Ltd, INSTICC, Lockheed Martin, NASA, Northrop Grumman, OASIS, Oracle, Red Hat, Saab Systems, Sparx Systems, THALES, The

Standish Group, Toshiba, Toyota Motor Corporation, W3 Consortium and many universities. [136]

The OMG also champions their Model Driven Architecture strategy [133], [137]. MDA provides a solid guiding methodology based on the use of UML to provide the platform independent model [17], and then by following through with the modern object methodologies for an efficient development and deployment of the target system.

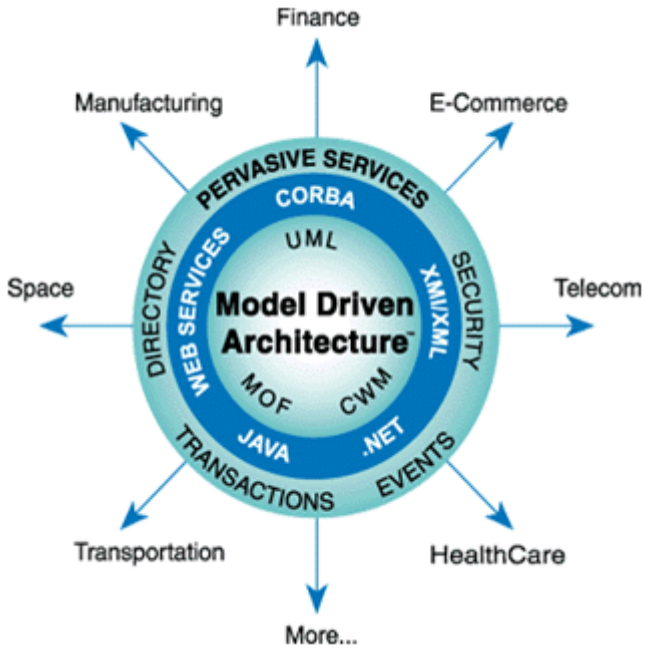


Figure 3 - OMG Model Driven Architecture [133]

MDA is based on the OMG's meta-modelling framework Meta Object Facility which defines rules to construct meta-models which provides the bindings to the Platform Independent Model and Platform Specific Model meta-models [138]. The OMG produced their MDA Guide [139] as a general overview of MDA and guide to OMG's architecture. This specification has now remained at v1.0.1 for over 10 years supporting some overall criticisms of OMG's overall direction.

By basing their specifications on such a well-known standard as UML, the OMG is well placed to influence the future development of UML based development and productivity tools such as Rational Rose [140] (see others listed in Table 1 - List of Popular MDE Tools).

The Unified Modelling Language [141] has been progressively developed into a precise toolset for the specification of system requirements [142], and is being provided with continually improving third party code generation options [143] [144]. The OMG is committed to a methodology based on the exclusive use of UML, which is continually being provided with candidate extensions for standards consideration [145] [146].

The latest version, UML 2.4.1, consists of the diagrams listed in Figure 4. Structure Diagrams show the static structure of the system and its parts on different abstraction and implementation levels and how those parts are related to each other - the elements in a structure diagram represent the meaningful concepts of a system, and may include abstract, real world and implementation concepts. Behaviour Diagrams show the dynamic behaviour of the objects in a system, which can be described as a series of changes to the system over time. [147]

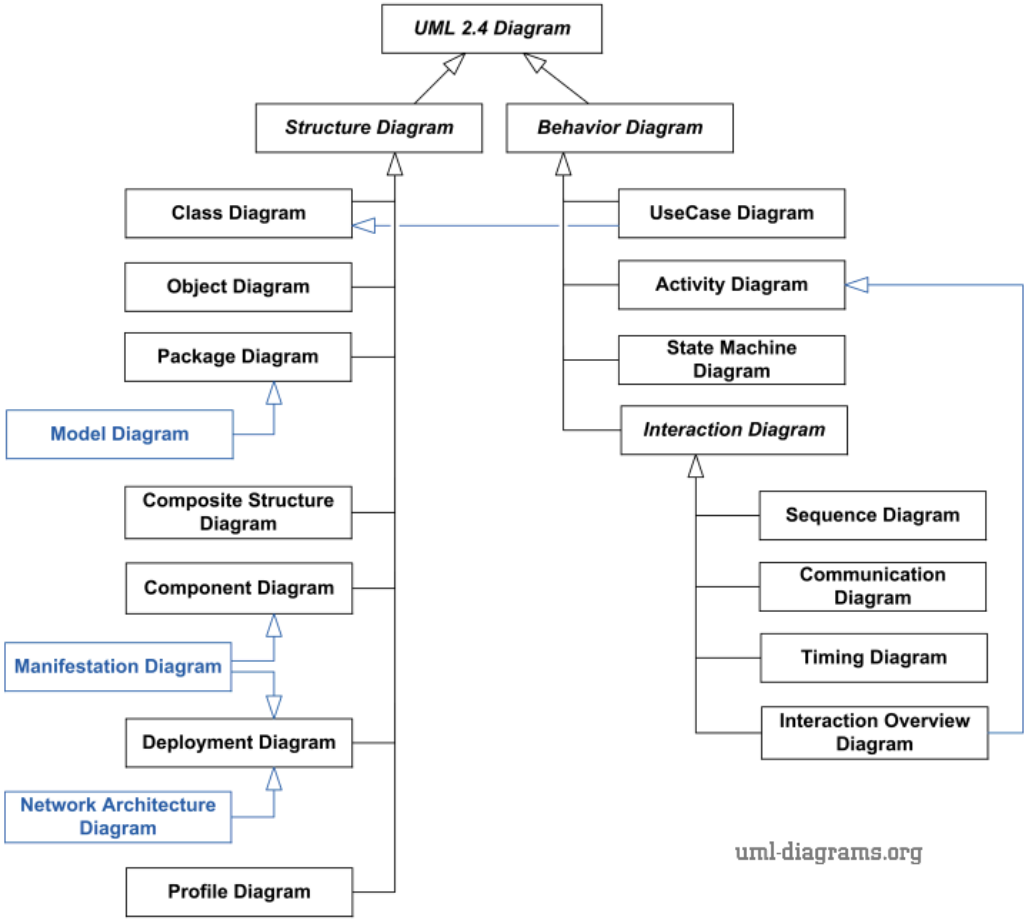


Figure 4 - UML 2.4 Diagrams

By using any one of a number of UML based tools it is possible to design and analyse a software system so that software requirements are met. Although UML's original purpose was for detailed software design and analysis, its extension mechanisms make it applicable to more broad problems. In the last few years, various extensions dealing with software architectures have been proposed. Software architectures can be specified using any number of different Architecture Description Languages (ADL) that are currently available.

[80] and [123] have shown how UML can be extended to fit their Chiron-2 (C2) architectural style. They propose to extend UML with new meta-class subclasses using the provided extension mechanisms. They show that sequence, collaboration and activity diagrams may be used to model their C2 architectural style's component behaviour.

While UML is a widely adopted standard for aiding software development it is a semi-formal language which lacks the precisely defined constructs to fully define application logic [44] and has been more commonly used as a coding accelerant rather than a coding replacement.

Criticisms of the OMG's MDA strategy include; MDA includes incomplete standards, vendor lock-in of MDA toolsets and the need of highly specialised skillsets. Additionally there are problems surrounding MDA tooling suchs as; scalability, generalised synthesis and extensibility [148]. Common problems with UML include; many diagrams and constructs are redundant or infrequently used, remaining ambiguities and inconsistencies, mismatches between the capabilities of UML and implementation languages and a dysfunctional XMI interchange format [122].

A further weakness of UML is that it has such extraordinary breadth and hence requires correspondingly high technical skills [149], requiring the difficult marriage of UML experts with business analysts and business process experts to obtain success in the commercial world. The current complexity of UML modelling is an inhibitor to adoption by non-technical staff who actually possess the business rules of an organisation [149] – currently restricting UML to indirect technical usage.

In order to bridge the crucial gap to align IT and business requires a less complex application design specification that could be achieved by either the development of new and simpler design metaphors or by using only a subset of UML features that can subsequently be used with far less technical expertise.

2.5.2 Eclipse Modeling Project

"Eclipse is a community for individuals and organizations who collaborate on commercially-friendly open source software. Its projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle. The Eclipse Foundation is a not-for-profit, member supported corporation that hosts the Eclipse projects and helps cultivate both an open source community and an ecosystem of complementary products and services" [150].

The Eclipse Foundation is modestly funded (~$4M annually) by almost 200 member organisations including; Ingres, Airbus, Arm, AT&T, Blackberry, CA Technologies, Cisco, Compuware, Dell, eBay, Google, Hewlett-Packard, IBM, IDG, Intel, Nokia, OMG, Oracle, Bosch, SAP, Siemens, Sony, Texas Instruments, THALES plus numerous universities [151].

"The Eclipse Modelling Project focuses on the evolution and promotion of model-based development technologies within the Eclipse community by providing a unified set of modelling frameworks, tooling, and standards implementations" [135].

The scope of EMP includes [152]:

- **Abstract Syntax Development**: a framework to support the definition of abstract syntax for modelling languages that support business, system, and software modelling, using an industry standard modelling facility or language. It will support editing, validating, testing, querying, and refactoring models created with the modelling facility. This includes the production of general-purpose modelling languages in addition to application domain specific models.
- **Concrete Syntax Development**: support for the production of textual and graphical concrete syntax for an abstract syntax, including both manual and generative approaches to the production of these i.e. editors for any Domain-Specific Language.
- **Model Transformation**: the transformation of models using a transformation definition and associated technologies.
- **Model to Text Generation**: text generation from a model, typically source code of some programming language, including the merger of user changes to generated output, along with support for patterns.

- **Industry Standards**: support of industry standards to enable their creation and maintenance within the Eclipse community. The following industry standards are within the scope of the Modelling project and are either supported by current modelling projects, or are anticipated to be supported in the future:
  - Object Management Group standards: Meta-Object Facility, Unified Modelling Language and UML Profiles not falling within the scope of other projects, Model-Driven Architecture related specifications, Query-View-Transformation (QVT), MOF to Text (MOF2T), Diagram Interchange Specification (DIS), XML Metadata Interchange.
  - Business Process Modelling Notation
  - Business Process Definition Metamodel (BPDM)
  - XML Schema Definition (XSD)
- **Domain-Specific Modelling**: support the emerging trend of Domain-Specific Languages. The generative production of editors for textual notations is an essential component of DSL support within Eclipse, and required if Eclipse is to be used as a "language workbench." The Modelling project will provide, within its scope, the generative aspect of producing these editors to complement graphical editors for a modelled domain.

All up the EMP consists of over 60 project components. About half of these are considered to be “Mature” which is classified as “the project team has demonstrated that they are an open-source project with an open and transparent process; an actively involved and growing community; and Eclipse Quality technology”. The remainder are in “Incubation” for which “the purpose of the incubation phase is to establish a fully-functioning open-source project. In this context, incubation is about developing the process, the community, and the technology”. [153]

None of the Eclipse components, including the EMP subset, are listed as “Top-Level” which is defined as “Projects that have demonstrated the characteristics of a Top-Level Project (e.g., consistent leadership in a technical area and the recruitment of a wider developer community) can be promoted to Top-Level Project status”. [135]. Further, just under half of the EMP components have been released as part of

the most recent general release [154], providing an indication that the project has a large amount of development remaining, albeit while providing ready access to its progressive releases.

Eclipse has been a major open source development in active use by thousands of developers worldwide. The Eclipse Modelling Project is a major initiative of the Eclipse Foundation and can be expected to develop high usage modelling toolsets, particularly for the UML2 specification.

It is the other Domain-Specific Modelling aspects of the EMP that interest us the most as their proposed generic modelling tools may be expected to present readily configurable templates to aid in the graphical modelling and background model translations and executions for ongoing development of runtime engines and editors for this thesis' proposed model and execute environment.

2.6 Other Application Development Issues

Developing application software, regardless of the project management and software development methodologies, design tools and programming languages utilised, is only the initial means to producing useable software.

Throughout the remainder of the applications effective lifecycle the application will need to go through periods of; defect management, patches, upgrades, platform changes and user reviews to maintain the applications effective business utilisation. These aspects typically attract a significant proportion of the application lifecycle cost and need to be managed effectively.

Significantly, the use of appropriate model based development and ongoing model management could be expected to streamline the effort and costs of these requirements.

2.6.1 Alignment of IT

A major issue that has typically plagued successful system implementations is a suitable alignment process of the solution providers (IT and third party developers) with the problem domain (the proposed users and process owners) [155].

Newer methodologies attempt to provide some alleviation by escalating user interaction within the development processes, although this benefit may be offset by the corresponding elevation in more complex technology requirements and widening

of the communication barrier between the technical and non-technical humans involved [156].

In a model based application environment end users would be expected to be engaged on a more frequent basis as a more Agile methodology can be used. Additionally higher end design and modelling tools that can be used by knowledgeable end users could be utilised to provide users with some direct development capability.

2.6.2 Application Flexibility

How much of an application's functionality is static and how much is dynamic? How accessible is the data schema and application functionality to third parties? These are key concerns that need to be addressed to significantly improve the current widespread organizational operational inefficiencies, software development duplication efforts and imposed restrictions on software capability which include:

Application Logic Openness: Software developers can and do readily lock access to the underlying application and effectively achieve closed systems via obfuscation, compiled software and proprietary code libraries and technologies. Model based applications can expose the application logic and workflow definitions within the model and allow users to ignore much of the underlying or supporting technologies.

Data Structure Availability: The internal data structures of an EIS system can become very complex, especially over multiple generations where legacy data structures have been maintained and where new technologies have been progressively added to the EIS functionality, often resulting in severe internal data segmentation. Model based applications define all data objects within the same consistent model so that all data is always well defined and readily available for access and processing via the same logic definition mechanisms.

Configurability vs. Customization: Software developers do not apply consistent capability or definitions of the level of flexibility that their applications deliver. Some may define that a configuration is based on an attribute definition change executed via a provided user interface feature, whilst others may extend the definition to supporting but requiring user developed application code to be created and integrated. The common aspect is that there is usually only a small subset of application features that can be readily modified in most applications as a configuration. Model based

applications allow every aspect of the defined application logic to be modified, from the simplest to the most complex user interface, workflow or data objects.

Personal Flexibility: EIS applications are often the core workflow mechanism for many users and user groups to perform their operational roles however the same core functionality of the EIS, whether customized or not, may not best suit all users or even user groups. Different logical workflows may be required to achieve maximum efficiency however it may not in a traditional software development sense be considered worthwhile to implement any personal or localized customizations. Model based applications with accessible model editors could provide the same (re)design capability to knowledgeable users allowing minor or major changes to be defined as is appropriate for any specific individual or user group need.

Organizational Flexibility: EIS applications tend to often provide only the most common functionality and rarely will the "out of the box" workflows suit all customers and all of their users. High levels of customization are commonplace for customers to achieve a suitable level of compatibility with their organisational workflows. Model based applications can't avoid this problem entirely but they can provide great flexibility, economy and speed of delivery through the use of accessible model editors rather than only pure code based customisations.

2.6.3 Configuration

Most applications will provide some level of user configuration, whereby some aspects of the application can be readily defined by authorized users that will modify some application behaviour. Users of business applications would be familiar with the types of application configuration provided in some applications such as: the ability for users to select their own colour scheme for aspects of the user interface; the option to set some environmental options e.g. international locale to adjust some items' display attributes; or save the screen positions of user positioned screens.

More advanced configuration features of some applications may also allow users to: create simple reports or user defined data extractions and save them as accessible visual objects; or create simple template based user entry screens based on a data table.

There is no common or minimum capability for which user configuration options are provided in applications. Some applications may provide extensive configuration options whilst others may provide minimal flexibility options. While varying in

complexity the generally available configurable content for end users tends to be limited to simplistic features [157] with application customization being required for more advanced features. [158] long ago identified the need to focus on more configurable software to benefit users and developers as a joint initiative of software development to merge configuration and customization aspects. [159] extends the configuration options to multi-tenanted SaaS systems.

As every aspect of a model based application is an object that should be configurable via accessible model editors (rather than via code) then every application feature from the simple to the complex should be optionally configured by authorized users and specified to apply to only specific users or user groups. Simple features will require only basic knowledge to configure whilst more complex features will require a necessarily deeper understanding of function and capability.

2.6.4 Customization

EIS applications generally consist of three common layers or at least conceptual considerations for development; user interface, business logic and the database repository. Traditional EIS application development almost exclusively requires highly trained developers fluent in the various and often multiple languages, protocols and technologies that constitute the EIS application that may have been progressively developed over many years, containing multiple legacy technologies at any time.

The scope of the typical end user to extend functionality changes beyond that permitted by the commonplace user configuration capabilities is typically quite minimal as traditional software customizations will require:

- Access to the entire or at least partial application source code,
- Documentation and knowledge of the Original Equipment Manufacturer's internal software structure, design, directory and Application Programming Interfaces,
- Appropriate software development licenses and editor environments, correctly configured to produce compatible output software,
- Required technical skills to implement the specific customizations.

These requirements effectively need to duplicate much of the software development environment of the OEM software developer which is always going to be complex and expensive to establish and maintain, and is exclusively the domain of the skilled technical programmer. Whether the OEM software developer is willing to

expose any level of its often proprietary intellectual property to third parties or customers is purely at the behest of the individual OEM.

All features of model based applications could be permitted to be customized by authorized end users, aiming potentially at the knowledgeable business user or power user rather than solely restricted to technical experts. Defining complex or new application segments will require a correspondingly higher understanding of model functionality which a technical programmer can certainly fulfil as could a knowledgeable power user.

In a traditional organization environment, usually only the highest priority customizations, if any, are likely to be implemented. A key aspect for model based applications is that these customization features are then available to be created by any authorized user, for any particular purpose, drastically increasing the scope for rapid organizational and personal workflow improvement.

2.6.5 Software Version Management

Version control is the goal of software configuration management, to ensure the controlled change or development of the software system [160], to track the development of the components and manage the baseline of software developments [161] including throughout the various phases of a project as [162] provides for.

In traditional software development the atomic level to which version control can be applied varies on the version control systems used but can be as high as individual source code files or database table definitions. The atomic level to apply version control for model based applications is each individual object's attribute definition within the application model. Model based version control needs to be managed at the lowest levels as it is also fundamentally tied to maintaining model integrity and permitting direct dynamic execution as [163] demonstrates with their graph-based conflict detection algorithms.

Version control in model based applications can be an automatic function closely related to the management and support of the temporal status of application model and data at any point in time. An associated technique for identifying changes between versions of software [164] is a classical key approach when applied to the underlying application model and combined with the runtime version control of distributed components of [165] is instrumental to potentially allowing an automated update approach to be applied to model based applications.

2.6.6 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 (Enterprise Resource Planning) style systems tend to utilize either the version update process or otherwise install the new version cleanly 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 due to factors such as; local site specific customisations conflict with the updates, local site delays in maintaining updates can require additional specific upgrade efforts, and sometimes less effort and quality assurance seems to be expended on producing each specific update program than on the primary software product [166], exacerbating existing common issues with system development quality assurance [167]. Managers of EIS upgrades can attest to the often extensive projects required for particularly major version EIS upgrades which can require months of effort and considerable expense.

Model based applications could be automatically updated as each update is a set of modified application model changes that were performed in a specific order to advance from one version of the application model to the next. There should be no need for specialist update applications beyond a generic model updater.

Updates might even be performed on a live operating application although prudence would suggest performing updates in an offline state as some updates may involve individually lengthy executions e.g. where schema changes occur in large data tables.

2.6.7 Speed to Deliver Customizations

EIS software developers have many considerations when candidate customizations are proposed; resource conflicts with their ongoing core development teams, a wide customer base generating many individual customization requests,

analysis of customizations for consideration as core application functionality inclusion, plus commercial engagement processes.

Customers themselves have many considerations such as prioritization of often many customization requests, budgetary constraints, internal agreement on scope and the ultimate scheduling of associated testing and updates. The net effect can cause significant delays from conception to eventual implementation, with prioritization and business case determinations excluding an often majority of individually smaller but overall significant set of candidate improvements.

With a more efficient and direct configuration and customisation capability model based applications via their accessible editors can not only significantly lower the individual cost of each logic definition change to allow a corresponding increase in overall scope but through de-centralization can drastically reduce the delivery time, as knowledgeable users can create their own logic changes, and more complex model logic changes can be outsourced to any suitable technical resource such as power users, third parties or the OEM - all as simultaneous asynchronous activities. Additionally the automated update capability of model based application will greatly speed the deployment of any model logic changes.

2.6.8 Effort and Expense

The natural market forces of competition do not often strongly apply when dealing with an OEM or restricted set of authorized third party developers which contributes to the typically high level of expense to develop and customise traditional EIS customizations.

Whilst it may be prudent to engage the most knowledgeable and technical resources of the OEM for the more complex of customizations, or logic changes in a model based application, there is a wide range of model logic changes that can be effectively performed locally by internal users, subject matter experts and technical resources. The ready use of appropriate internal resources would typically provide much lower cost and turnaround particularly for the often large number of potential model logic changes that may be individually minor and relatively trivial to implement but overall can be representative of a potentially major collective opportunity benefit.

2.6.9 Overall Lifecycle Costs

The initial implementation cost of an EIS application is typically only a small fraction of the overall system lifecycle cost. The size and scope of customizations vary with associated proportional costs to develop and implement. Every future EIS update from the OEM needs to be reviewed and tested for ongoing compatibility with all customizations and may often involve significant effort and re-work to ensure that the customizations will maintain compatibility with the new core application changes.

Model based applications should be expected to deliver much lower comparative lifecycle costs compared to traditional EIS applications due to; reduced initial development effort due to using pre-built functionality rather than coding every aspect, utilising knowledgeable end users to configure and customise aspects of the application model rather than often more expensive technical experts, reduced rework by utilising end users to a greater degree, allowing greater access to controlled model configuration and customisation providing faster improvement turnaround, and reducing patch, update and upgrade efforts with automated model updates rather than manual customised upgrade procedures.

2.6.10 Organization Efficiency

Due to the high lifecycle costs of customizations usually only the highest priority modifications are ever implemented. Accordingly there is an indirect organizational cost incurred for every identified but unfulfilled improvement due to continuing to operate with the identified inefficiencies of the current available workflows and processes.

Model based applications should reduce the overall lifecycle costs compared to traditional EIS applications providing increased profitability and scope for significant further model logic changes as may be desirable. Additionally similar comparative savings then apply to the creation of each model logic change making each modification cheaper to implement again providing further scope to fund further improvements.

The potential comparative floor price for many model logic changes can be even lower as simpler modifications may often be more directly implemented locally and immediately whereas even trivial customizations can often incur a minimum but proportionately higher level of cost and effort from external developers.

These key issues have been in existence for the history of large scale commercial software applications such as EIS style applications. I believe that their resolution, or at least significant improvement, requires a paradigm shift away from the traditional software development lifecycle, to model based applications with their simpler and more accessible model logic change capability as a major aspect to providing the required major improvements.

2.7 Evaluation Summary

All of the above mentioned methodologies, models and tools are useful in providing efficiencies to some parts of application software development, and in combination they can be used to good effect. However, none fully address providing a full development solution without the extensive assistance of dedicated technical specialists.

In summary I find that the primary weaknesses or inefficiencies in current EIS and major application software development are;

- **Project Management**: Depending on the type of software development the selection of an appropriate project management and system development methodology will at best optimise the development of the software but cannot necessarily (nor may even try to seek to) reduce the level of effort involved. However to ensure that a suitable software product is achieved it remains essential that an appropriate methodology should also be utilised to produce any model based application development.
- **Software and Technology**: Component based development, middleware and application frameworks are of direct benefit to developers in providing development efficiencies. The higher the level of reuse with direct integration will reduce application development - aiming for complete reuse with fully available functionality and automatic integration should minimise the overall effort.
- **Systems Development Tools**: The higher the level of abstraction away from program code the more readily accessible the abstraction designs are to trained designers. UML and associated technologies provide an increasingly powerful toolset for specification and partial (but increasing) generation of applications but require direct and ongoing technical expertise that is not

directly applicable to or useable by most end users. Design abstractions that are less technical and more relevant to the user base should bridge the technical gap and promote better understanding by and usage of such tools by knowledgeable end users.

- **Major Modelling Collaborations**: The current major modelling efforts, OMG and Eclipse, tend to be directed towards specialised technical skillsets, which will certainly increase coding and development efficiencies. More effort on model and execute is required to directly generate executable applications without the need for substantial technical coding resources.
- **Business Alignment and Real Productivity**: As almost all tools and technologies are aimed at software developer productivity there are minimal opportunities to bring a closer alignment of the business users with the IT system providers – maintaining the need for IT specialists to interpret and cater for the requirements of the business users. Providing more user accessible tools to allow the subject matter experts to capture their requirements and thus generate applications directly to meet these requirements must be the ultimate goal to achieve genuine major efficiency improvements.

This research is seeking a solution for the weaknesses mentioned above. The proposed solution incorporates a model of the required EIS application(s) driven by a framework of runtime components specifically for EIS applications that in conjunction with appropriate end user model definition tools for EIS applications constitutes a new approach for the rapid system development for EIS applications.

2.8 Conclusion

In this chapter I outline the current state of project management, software development tools, modelling tools and major industry modelling collaborations. The current work clearly shows significant progress in providing system development accelerants for mainly code based projects. The vast majority of modelling tools are also aimed at assisting code development or optimising only part of the application development solution.

I believe that the modelling must be complete, or end to end, rather than limited to any application or model subset, or individual application layer. Code generation is

generally useful for performance reasons as well as allowing modification by programmers but to become truly accessible to a wider audience, code generation (if done at all) needs to be 100% and fully automatic. The granularity of the application model also needs to provide for adequate customisation and configuration to appropriately refine the models behaviours.

A model and execute style solution is required to develop EIS style applications that can be used more directly by knowledgeable business users to directly capture their requirements and generate an application without the need for technical program coders.

In the next chapter I will outline the main problems to be solved and identify the key research issues for this thesis.

Chapter 3 - Problem Definition

3.1 Introduction

The world employs millions of computer programmers, writing millions of applications. Of these many applications there are likely thousands of different accounting programs, and human resource management programs, and inventory tracking systems, and customer relationship management programs – the vast majority of each type of major EIS or ERP application would perform a very similar role, with some functionality changes, operating on some different computer platforms, written for different languages.

Each program will require its own major software engineering effort over typically long lifecycles, involving many years of patches, upgrades and platform re-engineering projects. Many of the programs are commercial with many customers, some will be custom engineered for a specific user base. Overall it represents a massive effort involving hundreds of millions of person years from technical programmers and end users, with massive amounts of duplication, relying on a continual availability of highly specialised technical programmers. There must be a cheaper, faster way to produce software that is even more directly focussed.

The first chapter highlighted many of these duplicated efforts and the heavy reliance on specialised skillsets. Chapter 2 surveyed the literature and current state of industry software development tools and techniques. There are many model based

solutions and initiatives that are contributing to better programming efficiencies but few that seek to replace the role of technical programmers with fully featured models, toolsets and runtime engines managed instead by knowledgeable business users.

While there are some notable exceptions that are actively promoting the concept of model and execute they are still primarily relying on toolsets that can be used more efficiently by re-trained programmers. The issues addressed in this thesis extend model and execute further to make it accessible to business analysts, power users and even many end users.

In this chapter I will clearly outline the main problems to be solved; model accessibility and supporting model structures. I identify the primary research issues that address model and execute solutions for non-technical users. EIS functionality, user requirements, application deployment and flexible execution environments issues are discussed. In order to introduce these problems and issues, in the following sections, I will provide a definition of the key concepts that I have used throughout this thesis. These concepts and the subsequent documented solutions form the basis of the work presented in this thesis.

I will conclude this chapter by outlining the problem definition summary and research approach utilised throughout the thesis preparation.

3.2 Problem Overview

Most EIS style applications are major software engineering projects, whether they are created as commercial products or for internal use. The role of most end users will be to assist in establishing requirements in the earlier stages of a project and possibly to assist with testing during module creation and then with acceptance testing.

Using more modern agile project management will utilise end users as subject matter experts on a more continuous basis to directly assist technical coding teams as they capture requirements via short term prototyping sprints. This process can certainly help in ensuring a closer solution to unknown or evolving requirements but still requires the major efforts of technical programmers and other technical architectural skills.

Even EIS style applications that are produced with a heavy emphasis on CASE or modelling tools still largely tend to use these toolsets to model only partial aspects of a development or to generate the skeletons or templates of the application, still

requiring significant coding efforts to complete the application. Very few model and execute style tools exist that seek to drastically reduce the coding effort and none that aim to provide the tools and environment to completely remove the need for complex coding environments and encourage knowledgeable business end users to define and generate the EIS application.

As outlined in Chapter 1, our envisaged model based development would utilise openly accessible model structures from which the EIS applications would be directly generated or executed from.

These model structures could be largely defined by knowledgeable business users and subject matter experts such as business analysts, power users and even end users, without the strict need for technical coders – certainly, programmers can be of great assistance to define some complicated logic as they are professionals well suited to working with logical models, as well as helping integrating with third party systems – but we no longer want to rely on the need to directly code the applications.

Such a solution is not simple, and conforming to open models that make their data and functionality accessible to other third parties is likely a concerning proposition for many commercial organisations that may consider this a considerable risk to their intellectual property and business models, as customers would no longer necessarily remain "locked in" to their selected EIS application vendor.

In order to provide a solution architecture for this model based development, the following section will define the following key concepts: EIS/ERP applications, model and execute, model editor, automated runtime execution, service-oriented architecture and cloud accessibility and integration. These concepts form the basis of my proposed new model based solution.

3.3 Key Concepts

Model based development has different meanings depending on who is performing the modelling and what aspect of a system is actually being modelled. For example, it can be reasonable to state that any technical programmer is performing modelling as they are transforming some form of real world problem into a solution that is represented by or modelled by the source code that they have developed. So what type of modelling am I focussed on?

I have referred to EIS or ERP style applications as the candidate applications for this modelling. Why these specific applications? While complex and large in scope and functionality, they are strongly process driven yet tend to represent what might be considered as a technically simpler subset of applications, and hence a useful starting point to establish a solid modelling architecture. This core architecture could then be further extended as required to address more diverse problem domains.

3.3.1 EIS/ERP Applications

EIS and ERP style applications are large scale information systems that provide organisation wide access to key data and information services. For the purpose of this thesis any general reference to our targeted or modelled applications or, EIS or ERP applications, should be considered to refer to the following definition.

I define: *the class of Enterprise Information Systems applications as visual and interactive applications that prompt for the entry of appropriate transaction data and user events from the application end users, use rules based workflow sequences and actions and utilize database transactions in a (typically 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 applications such as; logistics, human resource, payroll, project costing, accounting, customer relationship management and other general database applications.*

These applications are strongly defined by their data schemas. Well defined data schemas can contribute directly to a sound understanding of much of the basic structure of a supporting application as well as many of the fundamental workflows and data transactions that occur. This information is readily gleaned from reverse engineering and analysis of the; entity relationships, entity and attribute names, types and stored data.

The primary nature of these applications is that data needs to be collected from either automated or manual sources, transactions processed against the collected data, and reports and other transactions to then be triggered in response to the transaction results.

These applications have not typically been visually challenging applications by which I mean that they have commonly utilised mainly text based interfaces, even

though these interfaces have mostly been updated to work within a modern Graphical User Interface (GUI).

They may now utilise more advanced GUI selection objects such as tree controls and generated graphical charting objects but usually little need for dynamic objects such as video or 3D editing, other than perhaps as a repository of source data.

The inclusion of more dynamic objects into the proposed application models is by no means intended to be permanently excluded, however as a practical starting point this candidate application definition alone represents a massive optimisation opportunity target. Future model extensions plus their supporting runtime components, catering for additional application types, would be expected to be able to be readily added to the core model structures once established.

3.3.2 Model and Execute

Model based development has different meanings depending on who is performing the modelling and what aspect of a system is actually being modelled. [168] defines a model as a *"thing used as an example to follow or imitate" or "simplified description, especially a mathematical one, of a system or process, to assist calculations and predictions".*

For example, it can be reasonable to state that any technical programmer is performing modelling as they are transforming some form of real world problem into a solution that is represented by or modelled by the source code that they have developed. Clearly, in this example their source code is indeed a model that is most commonly then further processed by a compiler to error check and transform the source model into binary code that will be executed directly by the target platform. This binary code is itself yet another model albeit in a low level form that would not be well understood by the originating source model programmer.

There are many possible model variants that could be generated in this way – for starters there are hundreds if not thousands of possible languages and technologies that could be chosen to provide a source model syntax, and once chosen, every programmer or group of programmers would likely produce a greatly varying source code base as the final model solution, for an identical problem – some programmers would provide extremely clear documentation and others descriptive models such as BPMN and UML diagrams, while others may provide virtually no supporting information about their solution.

It is this incredible diversity of potential solution code options, with the worst examples spawning the term “spaghetti code”, and the associated difficulty in clearly understanding much of the produced code and subsequently maintaining and modifying that code that has contributed to the need for development of higher order models that can be used to more clearly capture and define an application’s requirements.

So what type of modelling am I focussed on? Clearly higher level process modelling with visual representations such as the UML and BPMN are a preferred starting point. However, [169] summarises a major problem in modelling systems by “existing process modelling languages and especially executable process modelling languages are not designed for business users with-out programming knowledge.”

A modelling, with automated execution, environment that provides the visual clarity of good process modelling tools yet avoids exposing unnecessary complications of underlying technical architectures could provide business users with their own solution generation capability – minimising or eliminating the delays and translation errors that occur within the usual paradigm of; user to analyst, analyst to designer, designer to coder, coder to implementation etc.

3.3.3 Model Editor

Any source code that is used to capture and define the model of the requirements needs an efficient model editor. The majority of existing source code environments will be either compatible or provided with a code editor with which to manage the lifecycle of the source code. While the “user friendliness” of many source code editors might be considered low, this is often a subjective judgement based on a mismatch between an expected technical user’s level of expertise and the level of assistance provided by the editor. Certainly the major open source and commercial source code editors provide high levels of coding assistance suitable to aid even the earliest level technical programmer.

Models that rely on a higher degree of visual manipulation of model objects over the specification of source code text implement highly visual and interactive editor environments that match the underlying structure and syntax of their modelling environments.

The great majority of editors directly support source code development and are clearly aimed at the technical programmers as the key stakeholder and users of these

type of model environments. Also in firm agreement with the assessments of [169] where the structures, models and editors of process modelling languages are also not designed for business users.

In line with our requirement stated in the previous section, to provide business users with their own solution generation capability, a model editor is required to capture the requirements directly into a model format that can then be automatically executed from. Such an editor would necessarily be based upon a design metaphor that business users are already familiar with – an interactive execution environment of the application that they are designing.

By using such an interactive model editor, business users minimise the need to learn the underlying technical abstractions, focusing instead directly upon the visual outcomes matching their requirements as they iteratively refine and develop them. Such an editor will not just be limited to ensuring absolute syntactic integrity of the model but due to its interactive execution capability will directly provide a much higher degree of semantic verification of the requirements during model definition.

3.3.4 Automated Runtime Execution

The process of transforming a typical source code model into an executable format usually involves multiple iterations of editing the source code to first achieve syntactical compatibility whereby the source code model becomes in a format eligible for compilation. Following compilation, the semantics of the new executable binary model must be tested to verify that the new application segment is operating as expected.

Progressive advances in automated testing technologies have resulted in new capabilities to submit newly developed application segments to a high degree of operational verification greatly minimising the need for repetitive manual testing. However, the adoption of these automated testing technologies requires additional setup and strict adherence to development standards, limiting its rollout to those development teams with the appropriate budget and discipline. Unfortunately the penetration of these technologies remains relatively low to date, requiring the majority of semantic test efforts to be based on manual verification.

The use of an interactive model editor will prevent all syntactical compatibility issues as they will not be permitted to occur and thus verifies the fundamental

integrity of the defined model which provides the matching executable functionality for all modelled elements.

To fully empower business users with application generation this model needs to be the sole basis from which the application will perform execution. As syntactical verification has been guaranteed by the editor, there is no further impediment to immediate compilation and execution of the application from this model. Such compilation needs to be performed automatically and without technical direction from the business user application model definers. While direct compilation and execution in a traditional sense would be an option that would satisfy and empower business users, a more flexible approach would be for a runtime engine that asynchronously interpreted and executed the model on a model element basis as this would support greater interaction and flexibility with high model change rates by defining business users.

Semantic verification cannot be guaranteed with this approach but I believe that a much greater rate of semantic verification will be achieved, and faster, due to the interactive nature of the model editor – as the model editor will be capable of interactive execution capability to quickly identify and rectify semantic errors directly during the model capture and design.

3.3.5 Service-Oriented Architecture

From an application user's perspective the underlying architecture of the software application is not usually of prime importance – the key factor for them is that the software works when they want it to. To give business users maximum flexibility in; remote execution, personal customisation, and third party integration, the architecture of the runtime engine needs to be open, and to a fine grained level to expose low level functionality.

[170] defines *"a service is a unit of work performed by a service provider to achieve desired end results of a service consumer. A service is a function that is well-defined, self-contained, and does not depend on the context or state of other services"*.

Constructing the runtime engine on a service basis as potentially hundreds of separate components rather than as a separate large software construct will provide this required flexibility.

[171] defines Service-Oriented Architecture (SOA) as: *"SOA separates functions into distinct units or services, which developers make accessible over a network in order to allow users to combine and reuse them in the production of applications. These services communicate with each other by passing data from one service to another or by coordinating an activity between two or more services"*.

SOA encourages dividing larger applications into smaller discrete fine grained modules which are used to produce course grained services that can be easily integrated by others [172]. With this architecture services and clients can be changed independently, allowing developers to map distinct business processes as services that can be chained together in order to realize higher order collaborative behaviour.

3.3.6 Cloud Accessibility and Integration

In terms of this thesis there are two very different perspectives for the cloud; one for users of business software, and the second for the underlying software architecture of the runtime engine that executes the business software.

From the user perspective, the term "the cloud" is often just a metaphor for the Internet in general and users ideally need to be able to execute their business software from anyway, at any time, and interconnect one software system to another however they want – certainly an ideal situation but one that should be able to be progressively better accommodated. Certainly the ever increasing bandwidth availability and well selected user interface platforms can well satisfy the first two criteria.

There are a few dimensions to the software interconnection issue. The first is clearly the ability to integrate the functionality of one software system with another. This need has been partially serviced historically by specific and usually fixed integration points provided by software providing typically limited integration. The appropriate usage of SOA for this software solution and increasingly by other software systems will greatly enhance the overall integration potential. Certainly the internal application self-integration flexibility will be significant.

Another dimension is when there are multiple instances of the same software throughout distributed organisations. Data may often be duplicated and difficult to access often requiring much manual processing in order to collate individual data silos into clear and accurate hierarchical or corporate views. Resolving data and processes across such an instance distribution to better reflect holistic viewpoints should be more readily achievable.

From the architecture perspective, cloud computing involves distributed computing over a network, where a program or application may run on many connected computers at the same time, allowing users to access the application, store data, or perform any other computing task from anywhere in the world. [173] defines the benefits of such "computer outsourcing" as providing "great elasticity and scalability of resources. It minimizes client-side management overheads and benefit from a service provider's global expertise consolidation and bulk pricing, and helps users avoid the capital expense in acquiring computing resources".

Again the appropriate usage of SOA for this software solution will provide maximum flexibility in the deployment of the runtime engine components solution. The range of supported user access platforms accessing the runtime engine through the cloud will ultimately be based on the more commonly used technologies.

3.4 Research Issues

Applying the key concepts defined above I have identified the following four research issues for this thesis. These issues will be pursued in order to provide a solution to reduce the scale of technological barriers in EIS application software development and increase the openness of what many users experience as a closed or locked in application environment in the following chapters.

3.4.1 Research Issue 1: The Definition of an EIS Model Structure

The definition of a model structure that will adequately model the application features required in EIS applications encompassing the user interface, business logic workflow and transaction processing capability. Most design and modelling applications specialise on a specific tier or layer of the design - capturing all of the required model attributes within a single model allows the entire application to be considered.

Design efforts by organisations such as the OMG-MDA rely on progressive refinement of their models through separate stages such as the Computer Independent Model (CIM), then the Platform Independent Model and finally a Platform Specific Model which can make eminent sense when seeking to optimise a traditional development lifecycle and environment. However, as this thesis also seeks to pursue an alternative development lifecycle which further empowers business users, a further

simplification will be to merge much of the design into a single model where minimal platform related information is required as the runtime engine will flexibly perform this role to then execute the modelled application.

By initially considering EIS style applications which are a simplified subset of applications then a more realisable target is set to be achieved by a correspondingly simpler model than that of groups such as the OMG-MDA organisation. As they represent a wide range of industry domains they tend to tools that attempt to address the full range and scope, and necessarily requiring highly technical development staff fluent with these specific toolsets, rather than towards business users who already possess the domain knowledge and often already have the appropriate skills that would be needed to capture their design into an appropriately formatted model.

There is no obvious reason that would preclude this working EIS application model to be further enhanced with additional modelling elements to address many other application domains.

3.4.2 Research Issue 2: Design Accelerants for the Iterative Design of EIS Models

A key objective of this thesis is to shift the main effort of application development requirement for the EIS application logic from technical programmers to application users, with the greater business logic complexity emphasis on power users and business analysts.

This will necessarily change the basic application development lifecycle, potentially greatly simplifying it. The main expected benefits would be a merging of the analysis and design stages, with significant reduction of effort, followed by a virtually eliminated development stage as there would be no additional development required (assuming that the provided and supported modelled functionality was adequate), followed by a reduced testing stage (which only needs to test the modelled semantic logic rather than the usual additional testing of all syntactic logic), completing with a virtually eliminated deployment stage as the model updates can be deployed automatically.

Analogies will still exist for the modelling processes along with some unique aspects as facilitated by the use of common meta-data modelling. Defining application model meta-data broadly falls into a combination of the following:

- **Defining new meta-data**: creating new meta-data definitions for the modelled application,
- **Deriving the meta-data**: from some existing non meta-data EIS application (MDEIS) based objects such as reverse engineering from existing database schemas,
- **Editing existing meta-data**: to modify existing aspects of a model or extend the application logic,
- **Merging meta-data models**: where multiple meta-data EIS application models exist their meta-data models and thus application functionality can be merged.

As the application meta-data model is fundamentally structured data, the meta-data can even be hand loaded into the appropriate data structures for subsequent execution however this is hardly a user-friendly option. To efficiently manage the creation and maintenance of the meta-data models requires assistance editor software analogous to the Integrated Development Environments of traditional software development. In some areas this editor would be simpler than common IDEs but also necessarily include additional higher level features that need to be modelled as part of the meta-data EIS framework which may again be analogous to the various component add-ons that are available to common IDEs.

I will design accelerator mechanisms to expedite and simplify population of the model by business users, with user specified model data such as rules and relationships between application objects, wizards for defining model data entry sequences, user interface templates, external model reverse engineering and additional model objects that will facilitate integration between multiple models.

These accelerator constructs will be dependent on the final structures and workflows within the model although some aspects will be similar to other widely available reverse engineering functions.

A significant feature that will be considered is the issue of merging different application models which can be achieved by specifying nodes of commonality between the models and automatically executing the combined application model – such a simplified system of merging, sharing and integrating disparate applications is expected to provide significant benefits by reducing data duplication and workplace

repetition – analogous to a simple method of integrating existing disparate EIS applications this is a particularly unique and advantageous feature of the framework.

3.4.3 Research Issue 3: Design of a Prototype Agile Platform for Dynamic Execution

A domain specific model for EIS applications, whether in the traditional form as software source code or defined as the higher level meta-data EIS application model as described in this thesis, requires a separate execution environment that transforms the model into operational use.

In the traditional application development environment compilers are used to verify the syntax of the source code and produce an executable machine language file or a transitionally coded model file that will invoke the required functions of the full runtime environment as required during execution – the EIS application model similarly requires the support of its temporal runtime framework for execution.

The runtime engine for the meta-data EIS application model verifies the integrity of the defined model and provides the matching executable functionality for all modelled elements. The general requirement for any runtime engine is that full compatibility with and support for all features of the meta-data EIS application model is maintained, ensuring that the same model can be executed by any individually architected runtime engine and process the inputs to obtain identical outputs.

A key desirable aspect of the meta-data EIS application runtime engine is that it is able to dynamically respond to model changes i.e. the current meta-data EIS application model must be the source for the runtime engine and not require the often lengthy and convoluted compilation processes of traditional application development, nor their typically manually and delayed deployment of executables, particularly when customisations have been made for the end user.

This runtime engine is expected to be service based utilising any combination of technologies and deployment strategies. The high level design will document the key features and attributes of the runtime execution environment.

3.4.4 Research Issue 4: Definition of an Interface Language Specification for Cloud Access

The stored meta-data model is the entire basis for the definition and subsequent execution of the meta-data EIS applications. Much of the application logic workflow

will rely on the relationships and links between the visual objects defined as the user interface objects – in a wizard based editor environment much of these will be generated automatically based on the underlying data structures. However there will always be the need for additional logical processing to be performed beyond the limited capabilities of induction and deduction of the data schemas.

Additional command structures are required to communicate direct instructions to the meta-data EIS application runtime engine and its layers, to both define new meta-data components and to execute meta-data components in response to defined logic, requiring the definition of an interface language specification that could be used to access data and application services.

Any meta-data model editor will necessarily need to retrieve the meta-data from the model and display it to the logic definer using an appropriate presentation metaphor and design. As logic changes are defined and committed these editor based logic changes need to be translated into the appropriate formal syntax commands and submitted to the runtime engine as a structured meta-data model definition command. The runtime engine will process the logic change, committing if valid or otherwise rejecting.

During execution of a meta-data EIS application the runtime engine itself communicates between its execution components using the commands, invoking web service calls for remote components or systems whether instances of MDEIS applications or any technology suite framing the appropriate web service calls and security authorisations. The physical location and combinations of end users and distributed layer components of the runtime engine is immaterial – any widely distributed or cloud based execution is supported by the command structure subject to appropriate internetwork carriage and authorisation between the components.

Accordingly, any other programmatic interface can present correctly structured commands to interface to the runtime engine components from any legacy sources that can; provide the appropriate security credentials, formulate the correct commands, and communicate to the layer component via web services.

To facilitate the empowerment of business users these additional logic commands that can be defined within the model editor will be in a format that are readily understood by a huge user base - as a general Functions based syntax similar to that already globally employed in major spreadsheet application used by hundreds of millions of regular users. This is the target technology focus to meet business analysts,

power users and general application users rather than the highly trained technical specialist software programmers.

Global cloud access to the runtime engine instances including individual access to each object's methods and attributes is then provided by crafting web service calls to provide secure access to these core Functions from any source, including component separation of the runtime engines, or any external system interface.

3.5 Research Methods

Methodologies are the research process or philosophy that researchers follow to interpret their data to reach their conclusions [174]. These processes aim to apply scientific methods to solve complex tasks [175]. Research generally follows a process of "problem, hypothesis, analysis and argument" [176] - problems are identified, analysis occurs as proofs and developed solutions, forming the basis for the evaluation of the research outcomes.

In the field of information system research there are two primary categories of research approaches, namely; a Social Science approach, and a Science and Engineering approach.

Social Science research can be qualitative or quantitative research that often uses data acquisition methods such as survey or interview, with statistical or qualitative analysis of the research data to identify evidence that supports or refutes the formulated hypotheses [176], [177], [178].

Qualitative research often utilises in-depth interviews by the researcher to facilitate the investigation of issues of interest that may arise during the interview. It does not necessarily involve large data samples, nor is the gathered information necessarily formatted to readily support statistical analysis. Quantitative research typically involves large scale data gathering by means such as survey and statistical analysis of the collected data in order to prove or disprove the proposed formulated hypotheses [179]. A common social science research approach, through the use of survey forms and large survey populations, is to assist in the identification of underlying problems as a source for further investigation leading to the development of additional hypotheses.

This style of research assists researchers in their understanding of people, culture and social issues. [180] contend that the ability to understand researched phenomena

within a social and cultural context is compromised when the empirical data results are quantified. This kind of research can imply how well the methodology is accepted or not accepted and sometimes may be able to give the reason. Accordingly, in social science research, widespread acceptance of the research methodology is critical, encouraging research methodologies to be derived from existing and accepted methodologies - validation of the methodology is a strong determinator for acceptance of the research.

Science and Engineering research is concerned with confirming the stated theoretical predictions, encouraging the practice of resolving research by the production of a candidate solution [181]. This well considered approach consists of a generalised three tiered process model [176], [179] where.

- **Level 1 - Conceptual**: the researchers create and define new ideas and concepts supported by initial analysis.
- **Level 2 - Perceptual**: the researchers formulate new methodologies, methods, techniques and approaches through design and/or construction of the tools, services, environment or system, in part or fully, through implementation or determining implementation requirements.
- **Level 3 - Practical**: the researchers conduct test and validation of the research works through experimentation with case studies and real world examples, using laboratory or field testing methods.

Science and Engineering research may ultimately lead to new techniques, architectures, methodologies, devices or concepts that may combine to formulate new theoretical frameworks. This approach will often address not only the basis of the problems that need to be addressed but frequently also propose candidate solutions to those stated research problems.

In the next section, I will present my choice of the methodology that is to be used in this thesis.

3.6 Choice of Research Methods

This research and thesis deal with the development of a new methodology and paradigm for model based EIS software development. As such, it is obvious that this research is not Social Science research, and clearly falls into Science and Engineering research.

3.6.1 Research Method used in this Thesis

The concept of system design and development as a valid information systems research methodology requires that any proposed software system, model or framework must be designed and developed to test and measure the underlying concepts. All proposed concepts will go through a "concept, development, impact" research life-cycle before the proposed system serves as both a proof-of-concept and a foundation for future work. For this thesis, I will base the research approach on Nunamaker et al's 5-step approach [176] as a suitable methodology to address the research issues stated in 3.4 Research Issues.

In this approach to information systems design and development I centre on theory construction, experimentation and observation. Any problems or constraints that are discovered throughout the development process are used in a design feedback loop to progressively modify the solution concepts and theories as depicted in Figure 5 of Nunamaker et al's multimethodological approach to IS research.

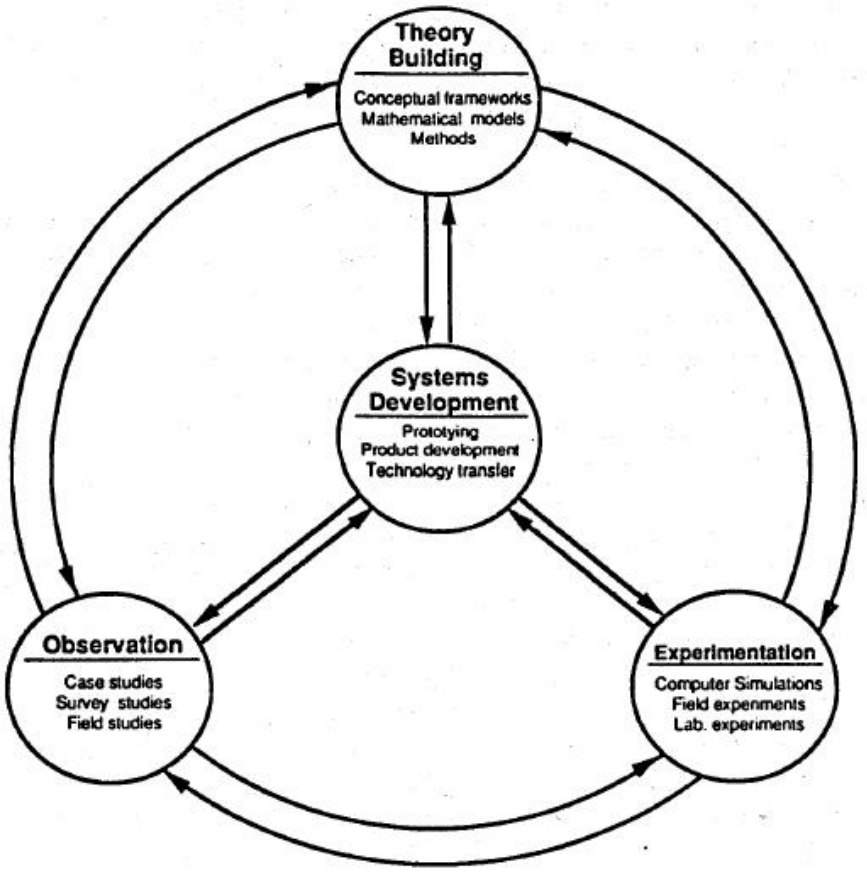


Figure 5 – Nunamaker et al's Multimethodological Approach to IS Research [176]

The 5-step approach addresses the following key steps below in Figure 6.

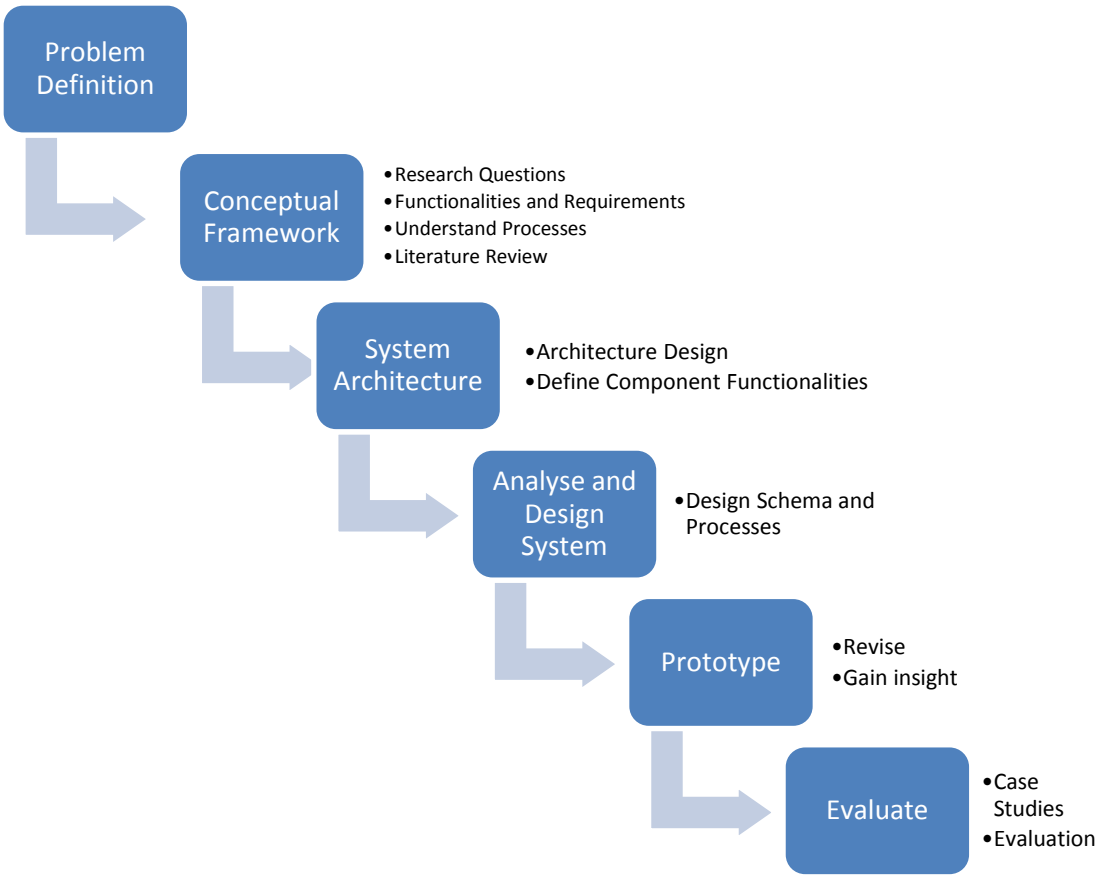


Figure 6 – Pictorial representation of the 6-Step Research Methodology used in this thesis

The following sections will address each of the key steps to be addressed throughout the research. They are extracted from and based on Nunamaker et al's key steps [176].

3.6.2 Problem Definition

The problem definition seeks to justify the significance of the questions that are being researched. It involves the analysis, interpretation, discussion, and evaluation of the originating issues and problems based on suitable criteria and balanced perspective.

3.6.3 Conceptual Framework

Researchers need to justify the significance of the research questions under pursuit. It requires significant study and understanding of the problem domain, the application of appropriate subject matter knowledge and experience to conceptually resolve the problems. When a proposed solution for the research problems cannot necessarily be proven mathematically, or where proposals are for new methods, researchers may elect to develop a demonstration of the validity of the solution, based on the new methods, techniques or design.

The conceptual framework is expected to lead to further theory building such as; declaring the "truth", formulating concepts, constructing methods and developing theories.

3.6.4 System Architecture

A system architecture provides a road map, putting the system components into perspective specifying system functionalities and defining structural relationships and dynamic interactions among system components.

Researchers must identify the constraints imposed by the environment, state the objectives and define the functionalities of the resulting system. Researchers will make assumptions about the research domain and technical environments, and state the system requirements under these constraints and assumptions, providing a design satisfying the requirements. Researchers should also emphasize any new functionalities or innovations of the proposed solution.

A proposed system's system architecture provide the instructions to build the system, decomposing into the components, their interactions plus a specification of the system functionalities. The design specification acts as a blueprint for future the implementation of the system.

3.6.5 Analyse and Design System

A research project's requirements may be driven by the new functionalities envisioned by the researchers. The design of a candidate solution is one of the most important parts of the system development process [182]. It requires understanding of the domain, the application or relevant technical knowledge and the synthesis and evaluation of alternate solutions.

A design should be based on theory and abstraction modelling. Design specifications should be used as blueprint for the system including design of; data structures, databases, program modules and functions.

3.6.6 Prototype

Building or designing a prototype system is a common engineering concept – its implementation or simulation demonstrates the feasibility of the design plus the usability and functionality of the system. Implementing a working system or simulation delivers essential feedback into the advantages and disadvantages of the proposed concepts, frameworks, and alternatives.

3.6.7 Evaluation

Once a system or simulation is built, researchers can test its usability as stated in the requirements definition, as well as observe potential impacts on individuals, groups or organisations. The evaluation results should be interpreted based on the conceptual framework and stated requirements.

System development is typically an evolutionary process. The resultant development feedback leads to continuing iterative development of the system, potentially including any discovery of newly observed phenomena. Empirical studies are performed on the developed system.

3.7 The Research Proposal

In this research, I have identified four key research issues that are resulting the complexity and flexibility problems that are associated with current and traditional EIS style application development. I propose the ultilisation of a model based framework, especially for enterprise application development with advanced model, framework, method and tools associate with it. Therefore, in this section I will give a high level proposal of the solution proposed that can address each research issue/

The proposed solution consists of five solutiion components, and each of these five solution components addresses each of the problems and issues associated with EIS style application development. This is then followed by evaluation the proposed solution.

Solution Component 1: To develop a conceptual framework for the capture of EIS application requirements into a user created model for automated execution and deployment:

This task focuses on the following aspects:

- Definition of an overall framework to cover all aspects of an EIS style application.
- Consider the additional operational requirements of a model based execution approach.
- Determine additional benefits that can be provided by a model based execution approach.
- Develop a detailed model to capture the requirements of an EIS style application.

I outline my conceptual framework, operational requirements and additional benefits in Chapter 4 - . Chapter 5 - outlines the detailed model developed to capture the EIS application and operational execution requirements.

Solution Component 2: To design a user accessible command structure that will provide ready access to all framework features supporting all logical and workflow oriented tasks. Chapter 8 - lists the additional reference commands available to the logic definers to provide advanced logic and workflow definition.

Solution Component 3: To design accelerant mechanisms for use by business users to enhance their ability to define model based EIS style applications, and minimise the current requirement for technical development staff. Chapter 7 - defines the required accelerants to be incorporated into a model editor environment.

Solution Component 4: To provide a high-level design of a prototype runtime environment to support the model framework in providing agile execution of the model objects. Chapter 6 - details the execution requirements of the runtime engine, architecture options, and the advanced options to be provided to cater for the unique advantages of the model based framework.

Solution Component 5: Demonstrate my proposed model based framework through a case study simulation of creating an EIS style application. I fully complete the logic definition requirements for all features and requirements of a commonly available and verifiable demonstration system, Microsoft's Northwind Order

Management System. This extensive logic definition exercise is detailed and reviewed in Chapter 9 - .

Solution Evaluation : To evaluate and validate the proposed model, framework, methodology and tools. I discuss the advantages and disadvantages of the proposed methods and conclude the research in Chapter 10 - .

The Reserch Approach to Solution Development

The research approach for the development of proposed solution follows five main steps: Construct models and a conceptual framework, design and implement the conceptual model and the framework the building the prototype system with new techniques and tools, followed by Analyse and evaluate the system. These steps have represented the choice of the methodology described earlier on in this Chapter.

The primary motivation for using the chosen research method is that this research aims to develop a improved conceptual model and framework with new techniques and tools to support automated EIS development. The research approach also include evaluation in order to 'prove' the concept.

The research approaches here defined how to address the four key issues. In earlier of this Chapter, I have identified four research issues that are aimed at solving the complexity and flexibility problems that are associated with current and traditional EIS style application development, by progressing to a model based framework with advanced model and framework as well as tools.

In this section I will give a brief discussion of the approach to solution development that can address each research issue:

- **The proposed solution for Research Issue 1**: EIS Model Structure – traditional development of EIS style software applications requires large scale effort by teams of technical experts, duplicating the efforts of hundreds of other commercial vendor teams and possibly that of thousands of private development teams. These efforts still most often result in quite fixed software configurations requiring often major efforts to subsequently modify. By capturing the business requirements into a model from which the application would then be directly executed could produce applications much faster with much greater flexibility.
- **The proposed solution for Research Issue 2**: Design Accelerants for the Model – all models require an efficient means of populating the model. As the fundamental business requirements is sourced from business subject matter experts, the EIS model editor should be primarily used directly by such business users, to simultaneously capture their requirements into an application design metaphor. These accelerants need to abstract much of the technical modelling away from these non-technical users through the

use of reverse engineering, wizards and templates for common workflows and user interactions.

- **The proposed solution for Research Issue 3**: Design Prototype Runtime Engine – the EIS model captures all of the application's logic requirements. A key framework accelerant is that the model is then used as the sole basis of the application execution, as performed by a runtime engine. I provide a high level design for the runtime engine including special features that can be uniquely provided by the use of a source model rather than fixed source code.
- **The proposed solution for Research Issue 4**: Cloud Access User Language – the stored EIS model is the logic source for the execution of the modelled application. While many features of an application can often be readily interpreted by reverse engineering and defined by specific wizard style accelerants, some business logic needs more fine detailed specification. To promote logic definition to business users rather than just technical experts I define a function based language similar to that used in major spreadsheet software, and already familiar to millions of business users. For remote and cloud based interfacing, integration and execution, these commands will be accessible by web services.

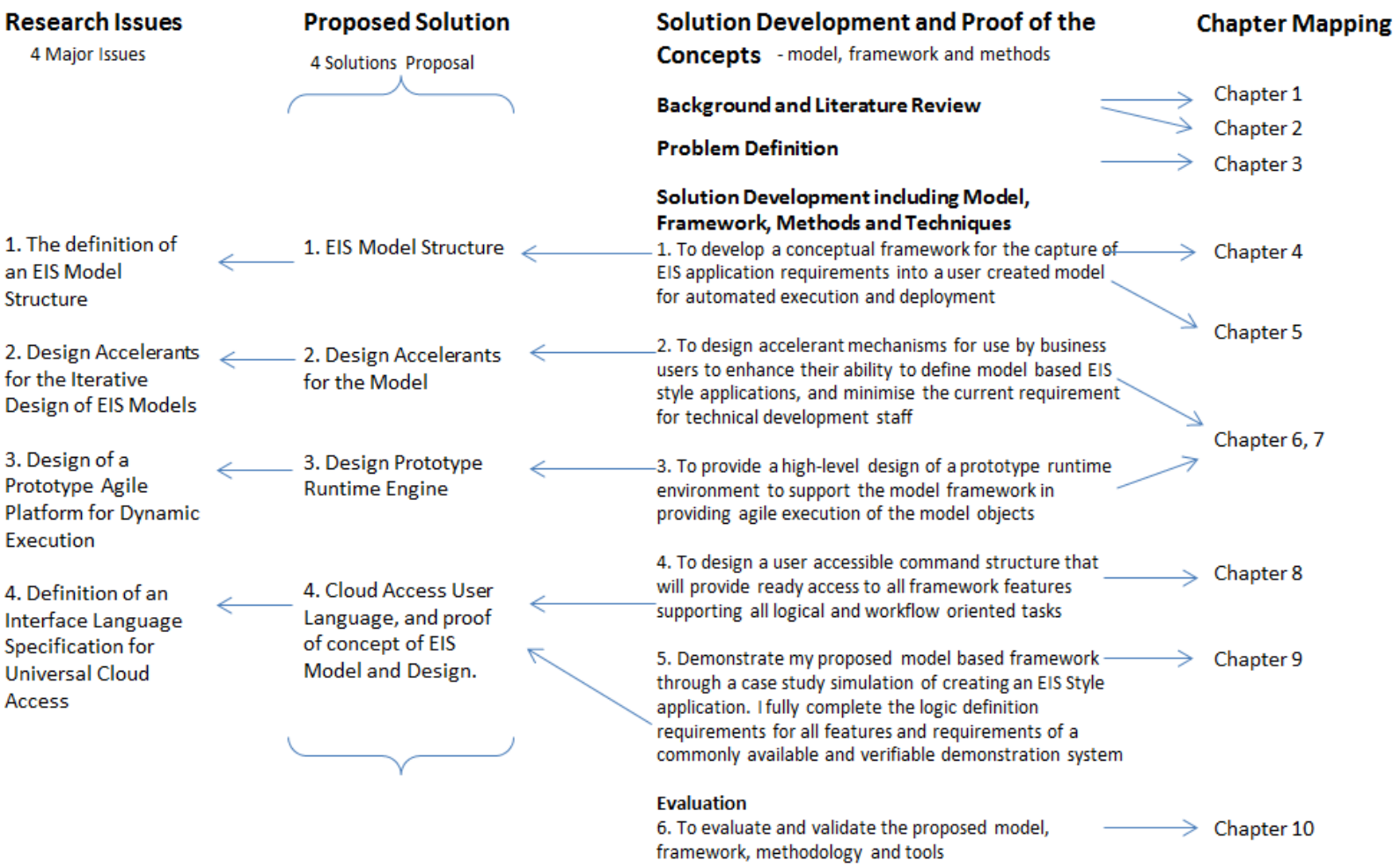


Figure 7 – The proposed solution development

Figure 7 above depicts the relationships between each of the Research Issues, Proposed Solution components or solution elements, primary solution development Tasks and cross-links the developed thesis chapters that contain the relevant supporting works.

The general problems, literature studies, problem statements and solution proposal are covered from Chapter 1 - to Chapter 3 - ,

The Conceptual Framework is described in first part of Chapter 4 - that defines specific requirements and expectation of EIS style applications and specifically of model based EIS style applications. The System Architecture step is also defined in Chapter 4 - .

The Analyse and Design System step then extends into detailed model specifications in Chapter 5 - . As the model is developed in a CASE modelling tool which therefore contains the full detail of all modelled objects (amounting to almost 2000 pages of extracted documentation) only higher level overview diagrams of the major objects and their functionalities are included in this (still rather lengthy) chapter. Chapter 6 - captures the design requirements of the prototype runtime execution engine and environment. Chapter 7 - defines wizards and model generation accelerants within a model editing environment including options for recursive self-generating editor. Chapter 8 - lists the syntax for the user accessible Functions that provide access to all logical and operations aspects of the model and framework.

The Prototype step is reflected throughout Chapter 5 - through Chapter 8 - based on the ongoing results of simulation including the major case studies described in Chapter 9 - .

The Evaluate step is primarily performed as an extensive case study implementation of a verifiable smaller EIS style application, with detailed descriptions of each model definition step in a simulated IDE editor in Chapter 9 - . A final overall evaluation of the thesis findings is presented in Chapter 10 - .

3.8 Conclusion

A major challenge is to produce a model for EIS style applications that will adequately cater for the majority of commonly used features which can often be described and implemented as high level objects, while also providing the flexibility to define other simple and complex objects. This requires a combination of known

high level objects that need to be accurately modelled, plus the ability to model atomic level objects and allow them to be readily combined in any logical manner.

While traditional programming languages can fundamentally be used to code and model virtually any complexity, our additional challenge is to allow similar flexibility to be defined and exercised by non-technical users. I strongly believe that building on the legacy of millions of existing business users who already comfortably utilise function based logic statements in their everyday business life through spreadsheet usage, that providing enhancements to this function based language can effectively provide substantive application model definition capability to these non-technical users. This could provide dramatic improvements in the speed and flexibility with which future EIS style applications are delivered and executed.

In this chapter I have provided a problem overview, defined the key thesis concepts, summarised four primary research issues and defined my research approach. These outlined issues need to be addressed to provide an alternative pathway to developing and maintaining EIS style applications.

Thus, the primary issues to be investigated in the following chapters are; EIS application model, model definition accelerants, runtime engine design and cloud based user access language.

In the next chapter, I provide the conceptual solution to the issues addressed in this chapter: the conceptual framework for model based EIS style application execution including an analysis of unique benefits that such a modelled approach can provide over traditional application development methodologies.

Chapter 4 - Conceptual Framework for Temporal Meta-Model for Enterprise Information Systems

4.1 Introduction

All computer source code that provides the basis for any software execution is in itself just a collated model for the desired functionality of the software program – more often than not, a model that has been largely hand-coded by a highly trained computer programmer. The apparent complexity of the model in these cases is dependent on the relative fluency that the observer or user has with the language used for the source code, as well as the particular technology, generation and level of abstraction that the language is based on.

Thus, an experienced programmer fluent in the specific software modelling language may have little difficulty in reviewing, creating and modifying such software models, however, the non-programmer but computer savvy business user would typically have very little understanding of or capability with such source code based models.

This thesis aims to define the tools required to provide non-programmers with the capability to readily design, define and create meta-data EIS applications with comparative ease, speed and simplicity.

The success of a structured meta-data solution to modelling EIS applications will be largely reliant on the functionality that can be provided as defined by the higher level components of a supporting framework in tandem with the associated efficiencies gained by the corresponding reduction in continual individual duplication of components, and other inefficiencies identified in this chapter. This approach is not exclusive of the need for access to lower level components but for meta-data EIS applications it is feature set and the ready availability of the higher level functions in the framework that will provide the expected major efficiencies.

To achieve the additional objective of aiming for business user level staff to become responsible for defining the meta-data models, which will then directly become the executing application, will require the functionality of the higher level framework components to be exposed in a suitable and user friendly design metaphor.

There will of course be trade-offs between the absolute freedom of flexibility and function that can be provided by an expert programmer using the computer language of their choice, with the necessarily simplified functionality and workflow that can be readily defined in such a higher level modelling environment. However, it is the very nature of EIS applications that lends itself towards this style of solution; highly visual and interactive nature of the applications, reliant on the entry of appropriate data by the application users, and they are heavily biased towards rules based responses and database transactions as the appropriate action.

In the following sections of this chapter, I present an overview of the solution to these issues. The structured meta-data approach combined with the appropriate framework components and design metaphor is the basis for a greatly simplified and inexpensive generation of meta-data EIS applications throughout the majority of the common phases of the typical software development lifecycle. Through its concentration on business functionality rather than any technical infrastructure concerns, the framework is a model that is clearly focussed on the business user and immediacy of results rather than the software programmer and extended development timeframes.

4.2 Preliminary Concepts Definitions

In order to progress with readily understanding the context and functionality of the proposed meta-data framework solution the following definitions are provided as used throughout this thesis.

4.2.1 Framework Definition

Traditional dictionaries define a framework as "a structure for supporting or enclosing something else" or "a fundamental structure" or perhaps closer in the context of this thesis as "a set of assumptions, concepts, values, and practices that constitutes a way of viewing reality" (1).

The definition of a framework as described in this thesis consists of elements from each of the above definitions although the latter is more descriptive. The framework referred to includes the model structure for the EIS applications functionality and the execution engine to automatically run the model as a standard computer application.

The framework's set of assumptions includes the core assumption that its model is adequate for at least the fundamental EIS functionality. The framework's concepts include the core model of the EIS applications functionality, plus the required functionality of the execution engine that will then support the execution of the EIS applications as modelled. The framework's values are based on the imperative that the business processes of the required EIS applications can be modelled by business level knowledgeable users rather than specially trained IT development experts – of course the IT developers are still required to develop and maintain the core software behind the execution engine and modelling database. The framework's practices indicate that a substantially simpler application development lifecycle can now be followed, one where the Analysis phase is still required, however with a much shorter Design phase due to the design being captured as the model, and then replacing or eliminating the majority of the subsequent Development, Testing and Deployment phases due to the immediate usage and availability of, and execution by, the then available execution engine. Subsequent Maintenance lifecycle loops follow an identical repeated optimised path.

The reality that is viewed via this framework is the standard computer application output that business users would then interact with, providing them with the required business functionality as modelled.

4.2.2 Temporal Definition

Temporal is also commonly defined as “of or pertaining to time” or “pertaining to or concerned with the present life” or as will be seen of more relevance to the context of this thesis as “enduring for a time only; temporary; transitory (opposed to eternal)” (2).

The definition of temporal as described in this thesis refers to the use of the dimension of time as a key aspect of the data model and framework. Thus, data should not be considered static and enduring, rather that it has a timeline of existence for which it is true and considered to exist, providing amongst other benefits, a true audit trail and history of all transactions and changes that may occur to any stored data. i.e. the capability to access the meta-data EIS application at any date and time, and to review the data and operate the application with the data current and correct as it was at that specified historical date and time.

Of particular note in this thesis is that the required EIS application functionality is stored as a model as data or as meta-data (which will be defined shortly). Hence the identical data management processes that can provide for an enduring data timeline can also be applied to the model data (or meta-data) to provide an enduring application timeline i.e. allowing different and historical versions of an applications model to be executed directly. To clarify, even though application updates and version upgrades often “break” or produce historical incompatibilities in some application features and/or historical data, the meta-data EIS application is also capable of correctly time and version synchronising to provide the full and correct EIS functionality for any historical date and time with the appropriate data from that exact date and time.

4.2.3 Meta-Model Definition

The term meta-model has common definitions such as “a pragmatic communications model used to specify information in a speaker's language” (3), “a data model that specifies one or more other data models” (4), and “a model that defines the components of a conceptual model, process, or system” (5), however each of these definitions contributes to the whole as defined in the context of this theses.

The definition of a meta-model as described in this thesis refers to the various abstractions and data structures defined to model and store both the required business functionality and the additional information required for the automatic execution of

that business functionality as the final output as the meta-data EIS computer application.

The first common definition above can also refer to this thesis's meta-model which supports the communications aspects of both multi-lingual and internationalisation, as well as seeking to transcend the often difficult boundaries between the language of the business user and the IT developer. The second definition supports that this thesis's meta-model is multi-facetted in that there are different models and levels of models to allow for the direct execution of the meta-data as an EIS computer application, comprising the pure business requirements models of data and data processing, with the more application based models of user interfaces and workflow processing. The final definition encompasses the overall objectives of this thesis in defining the models, processes and system requirements of such a solution.

4.2.4 Enterprise Information Systems Definition

An Enterprise Information System is commonly considered to refer to computing systems that are of "enterprise class", providing a "high quality of service, dealing with large volumes of data and capable of supporting some large organisation" (6). A useful extension to this definition is provided as "applications that comprise an enterprise's existing system for handling company-wide information. These applications provide an information infrastructure for an enterprise" (7).

The definition of an Enterprise Information System as described in this thesis also refers more generally to transaction data driven information systems that have appropriate functionality for personal use, through to small to medium business use, and extending up to usage in the enterprise organisation space. Accordingly the term Enterprise Information System is used interchangeably in this thesis with an Information System.

This EIS objective is based on the highly structured nature of such transaction based applications that I summarise as highly 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 to process the data, and utilise database transactions in a relational database environment for storage and to complete the transactions – as these applications are typically structurally repetitive they tend to be a technically simpler subset of possible computer applications. They

consist of applications such as logistics, human resources, payroll, project costing, accounting and other general database applications.

More complex and diverse applications such as design and drawing software, spreadsheets and office applications, and hardware specific utilities such as DVD burning software would not be considered as Enterprise Information System applications although it is possible that future analysis of the feature sets of these more complex applications could be developed as extensions to the outcomes of this research.

4.2.5 Enterprise Architecture and Enterprise Information Systems

[183] defines Enterprise Architecture as "a conceptual blueprint that defines the structure and operation of an organization". Gartner's definition [184] is more comprehensive as "a discipline for proactively and holistically leading enterprise responses to disruptive forces by identifying and analysing the execution of change toward desired business vision and outcomes. EA delivers value by presenting business and IT leaders with signature-ready recommendations for adjusting policies and projects to achieve target business outcomes that capitalize on relevant business disruptions. EA is used to steer decision making toward the evolution of the future state architecture" as it captures the key elements of intent, purpose and delivery.

[185] extends upon the delivery aspect of EA to include the role of the key practitioners of enterprise architecture, Enterprise Architects: "This new paradigm in enterprise systems development and integration highlights the demand for enterprise architects who can understand and align business goals with a technical strategy and architecture capable of supporting current and future needs". As will become evident later in this thesis, Enterprise Architects would have a major role as Logic Definers in a meta-data based EIS application environment.

Obviously there is a clear link between the need for effective EA in organisations and the role of EIS applications as key contributing components of the overall EA architecture. Delivering upon the promise of effective EIS delivery as this thesis proposes can have significant positive flow-on effects for organisations' EA strategies.

4.3 Innovation of the Temporal Meta-Data Framework for Enterprise Information Systems

The following innovations have been identified as part of this research. Each innovation is described briefly in the following section with more detailed analysis of the functionality and benefits of the innovations provided further in this chapter and thesis.

4.3.1 Temporal Meta-Model Framework

Temporal data management is a well understood field as it applies to the common database. Whilst embedded temporal solutions have never become standardised in the major SQL database vendors, individualised solutions offering varying levels of complexity and functionality are relatively straightforward to implement by database developers.

The majority of effort for Model Driven Architecture solutions is naturally around developing the supporting technology and architectures. Whilst version management of the models has received some attention it tends to follow a similar paradigm that is applied to current source code management although then more generally applied to XML model segments. This innovation refers to how the application of temporal data management techniques to all of the individual atomic meta-data elements of the models can provide for a complete temporal execution of meta-data EIS applications by maintaining a perfect synchronisation of the historical data with the historical application versions and states.

Such a solution would minimise the reduction of information accessibility currently currently experienced in most EIS applications due to the need for internal data archival or rollup, as application functionality is changed due to often irreversible version upgrades. See 0

Summary of Enhanced Features Provided by the Temporal Meta-Data Framework for additional benefits.4.8.1.2.

4.3.2 Temporal Meta-Data Model for All Application Layers

EIS applications are typically well structured applications that I have summarised as highly 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 utilise database transactions in a relational database environment to complete the actions. They therefore are comprised of the three common layers of information system applications; the user interface, the business logic layer, and the database.

Many solutions provide models and abstraction for one or more of these layers but for overall system lifecycle efficiency the model should provide for all of these application layers which can drastically reduce the effort involved in application development and layer integration. The temporal meta-data model described in this thesis provides for all three of these application layers to provide a complete foundation for full application delivery and execution.

The innovation provided in this thesis is the definition of the meta-data suitable to model all aspects of the EIS applications' design requirements stored and available in the structured meta-data, which will form the basis for EIS application execution with the availability of the framework's runtime components. See 4.8.1.2 Temporal Meta-Data Management for additional details.

4.3.3 Automatic Application Execution of the Temporal Meta-Data Model

With the collective application design requirements stored and available in the structured meta-data format, those meta-data EIS applications can be executed automatically with the availability of the framework's runtime components, providing effectively instantly available execution of the defined meta-data EIS applications.

The effect of this innovation is that the widespread introduction of such an approach has the potential to drastically reduce the time to develop and deploy new generations of meta-data EIS applications. Effectively, once the analysis and design efforts have been completed the system would become available for immediate use! The virtual elimination of the coding, combined with the minimisation of the testing

and deployment stages has significant benefits for both the developer and the end users.

4.3.4 Instant Interaction EIS System Builder

Common core system development methodologies such as the Waterfall, Spiral, Fountain and V models have not been fundamentally altered as a result of modern technologies and in general we are still maintaining a similar paradigm for system development; analysis, design, develop code, test and deploy. New system development methodologies such as Prototyping, Agile Processes, Big Ball of Mud typically propose differing levels of task decompositions, parallelism, customer interaction etc and certainly do provide specific advantages when dutifully employed but they are not guaranteed to necessarily change the magnitude of the total effort.

An EIS must necessarily start with a review of its requirements and the preparation of a design. This innovation proposes that performance of the analysis combined with an efficient collection of this information can also perform the bulk of the design phase, largely as a simultaneous activity. Hence the two steps may be merged in our proposed new methodology.

Coupled with the framework's runtime engine, the innovation is that the meta-data will provide immediate execution of its current state. As the integrity of the model is maintained and guaranteed during definition of the model at each discrete and individual iterative step of the meta model definition process, model execution can proceed from any step, thus providing an instant feedback as to the desired functionality of each application change whether produced as the result of a single or batch of application definition changes. See Chapter 5 - Instant Interaction EIS System Modeller for full details of the model.

4.3.5 Instant User Customisation

At the most controlled level of defining the application meta-data, all meta-data is defined at the system level or as "core meta-data". Core meta-data is considered to be the highest level of meta-data in terms of over-riding priority and security by the meta-data management and framework runtime execution system.

Meta-data management processes can define lower priority meta-data that can be defined within a customisable but pre-defined umbrella of application meta-data scope which effectively extends the overall meta-data definition and thus the

functionality of the application by readily allowing other vendors and users to define their own add-on meta-data, hence application customisation, instantly and without coding.

This meta-data management capability provides an innovation that is analogous to a user or application customisation capability that can be deployed on the basis of core meta-data applications with pre-defined core application functionality, yet optionally allowing hierarchies of additional customisations to be applied, effectively supporting a global infrastructure of core meta-data applications provided and maintained by application and meta-data experts yet allowing for local modification for local conditions and business rules by local users, with the same benefits of managed instant modification and feedback in a secure environment. 5.4.5 Variant Logic describes the model detail for these aspects.

4.3.6 Global Application Access and Sharing via the Cloud

The physical location of the meta-data definitions and of the framework's runtime components for automated execution can be maintained independently of the user organisation. By invoking the application functionality via secure and standardised communication procedures, full advantage of the Internet Cloud can be obtained to provide global access to an organisation's potentially global user and customer base.

The standardisation and atomic nature of application functionality procedure calls also facilitates the interoperability of application functionality between applications, components, web pages and organisations operating in the Cloud, providing the innovative opportunity for a global interconnection and sharing of availability of processed data sources and processing capability as offered by the defined meta-data EIS functionality of the host application providers. This can provide a level of application and data interactivity that is no longer dependent on the vagaries and complexities of integrating different and multiple technologies and protocols between each host / customer pair on a global scale. Chapter 8 - Universal Access to Temporal Meta-Data Framework for EIS in the Cloud details the external interfaces to the meta-data EIS application.

The modelling capability and execution features of the temporal meta-model framework would expect to allow for the ranges of EIS/ERP applications listed below in Figure 8.

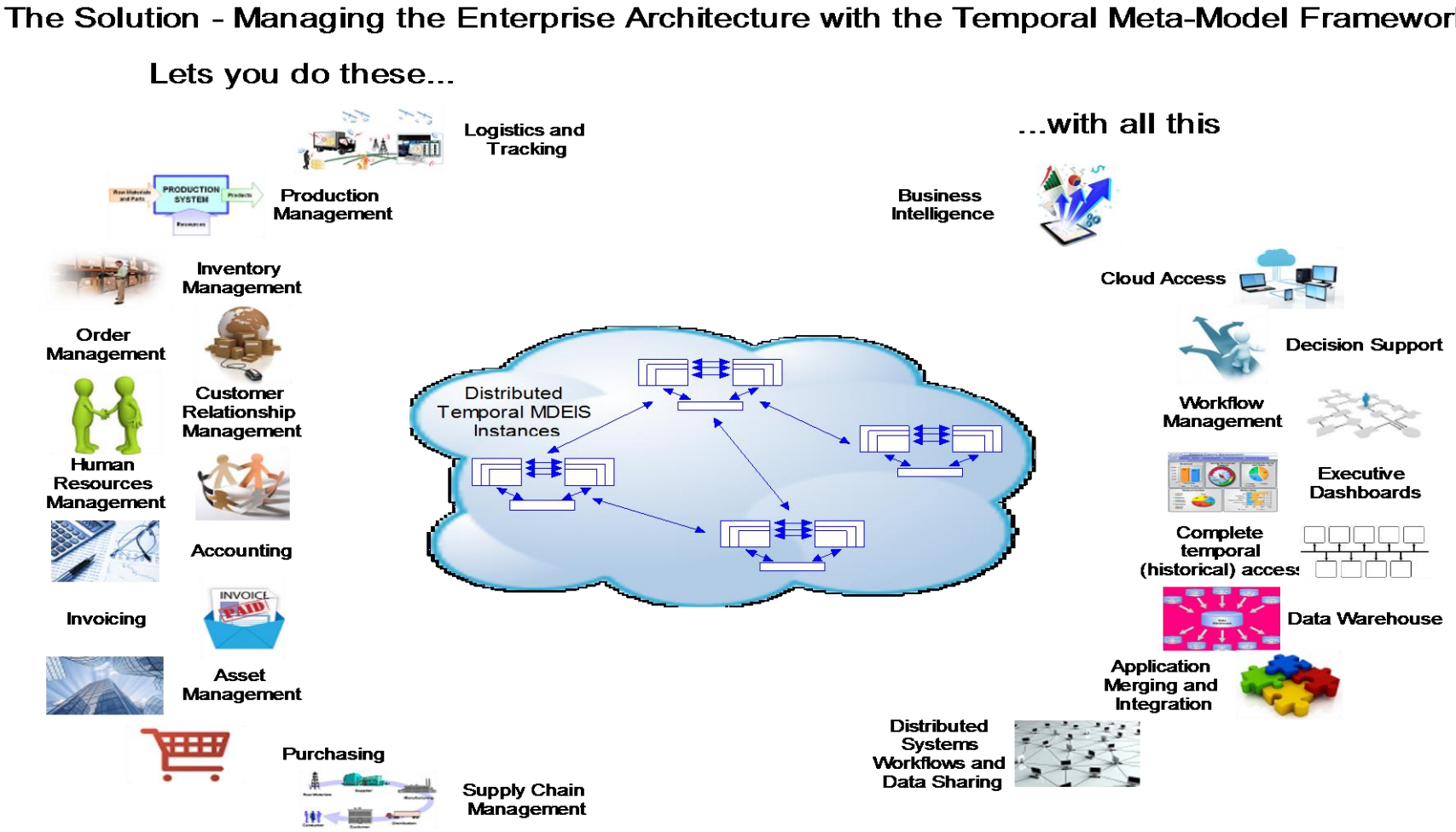


Figure 8 – Candidate EIS/ERP application systems and key framework functionality

4.4 Overview of the Temporal Meta-Data Framework

The success of a structured meta-data solution to modelling and executing EIS applications will be largely reliant on the functionality that can be provided as defined by the higher level components of the temporal meta-data framework in tandem with the associated efficiencies gained by the corresponding reduction in continual individual duplication of components. This approach is not exclusive of the need for access to lower level components but for meta-data EIS applications it is the feature set and ready availability of the higher level functions in the framework that will provide the expected major efficiencies.

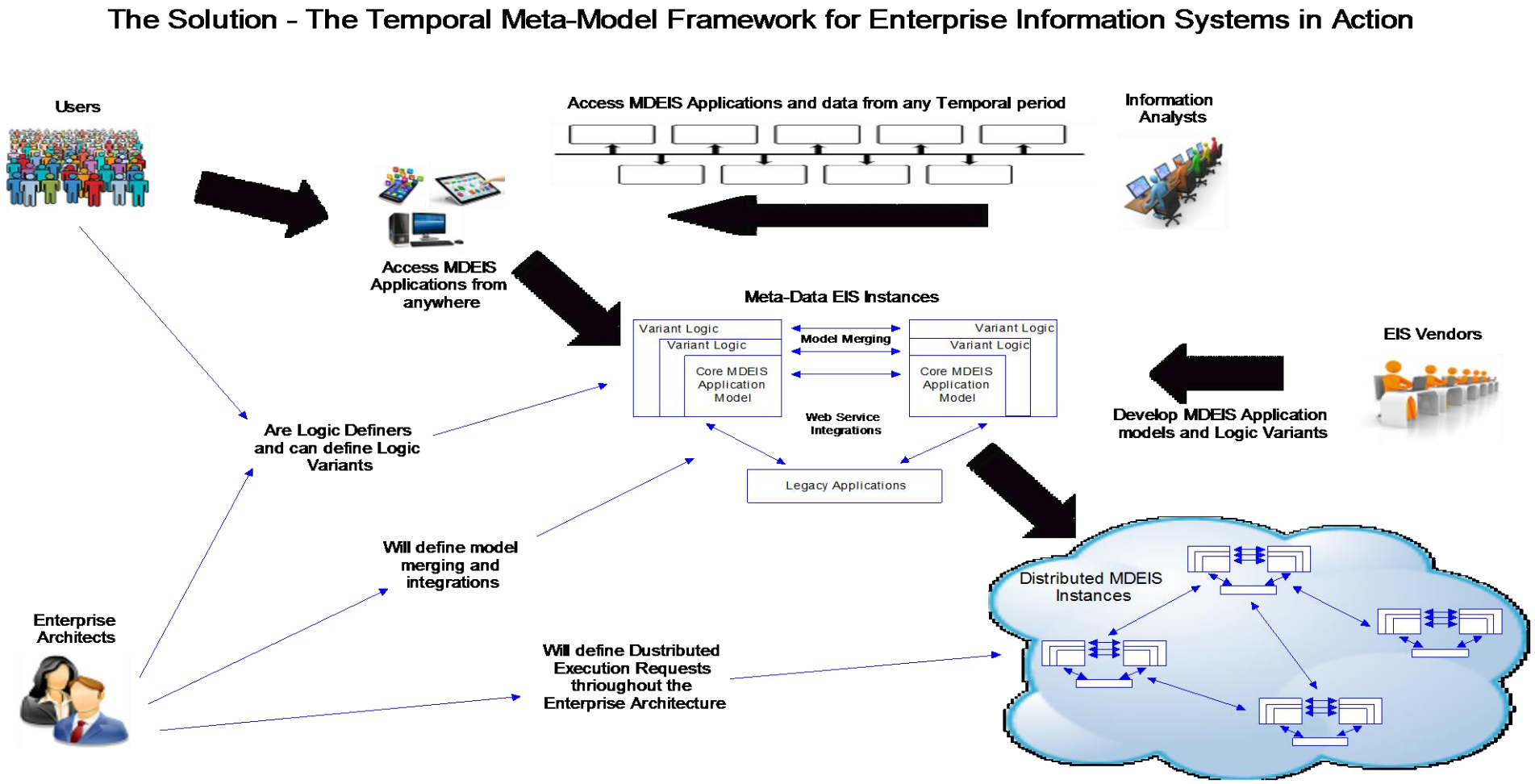


Figure 9 – Overview of the Temporal Meta-Model Framework for EIS in action

Figure 9 illustrates an operational environment for the temporal meta-model framework in operation, highlighting the use cases of the primary roles; vendors who create EIS models and Logic Variants, information analysts who may require access to the temporal application and data states throughout the application's history, users of the applications (who may also create Logic Variants), and enterprise architects who would establish and maintain (as key and empower Logic Definers) the integration, model merges and distributed integrations.

Figure 10 below provides a simple overview of the execution layers of the temporal meta-model framework – as a processing system in abstraction it is common to most EIS style applications which it will need to emulate, however as Figure 11 will start to show, the model-based functionality requires significant architectural differences to execute from the application models.

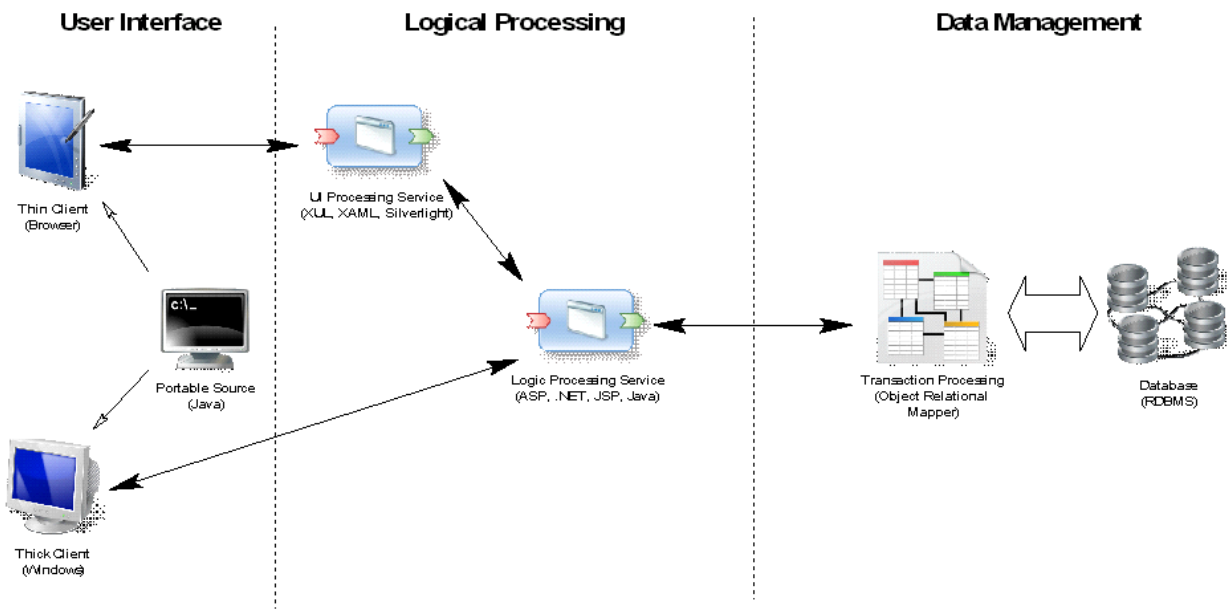


Figure 10 – Generalised Runtime Engine Architecture

An inherent feature of EIS applications is the highly visual and interactive nature of the applications. The success of EIS applications is reliant on the entry of appropriate data by the application users, and they are heavily biased towards rules based responses and database transactions as the appropriate action.

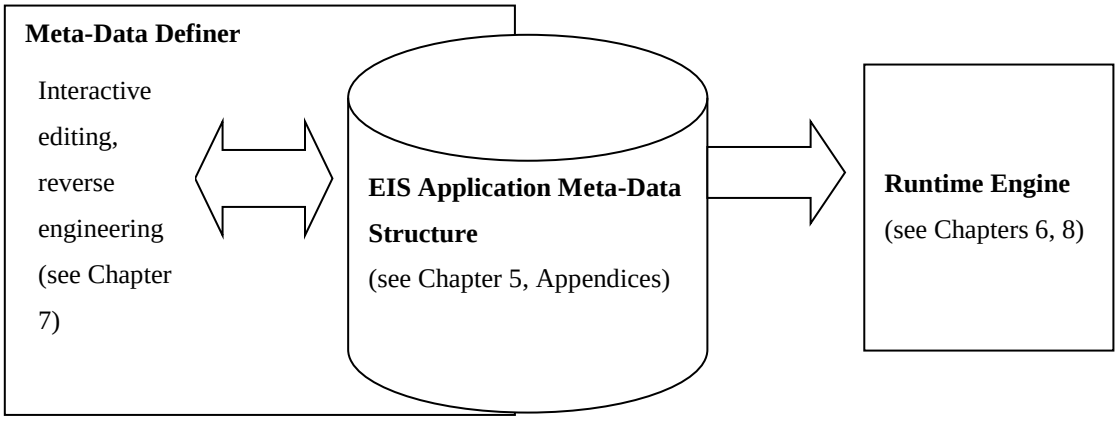


Figure 11 – Overview of the Basic Architecture of the Framework

Figure 11 above represents the highest level overview of the temporal meta-data framework referencing the chapters for further detailed information. It illustrates the high level flow of meta-data during the interactive editing process followed by the interpretation and execution of the meta-data by the runtime execution engine.

Figure 12 below presents a summary of the high-level workflows. The meta-data model is defined or modified, in a variety of ways which shall shortly be expanded upon, and provided to the framework's runtime execution engine for managed and automated update into the meta-data model. This may be via interactive editing sessions or as batch update (which are streams of sequential meta-data changes) to be applied.

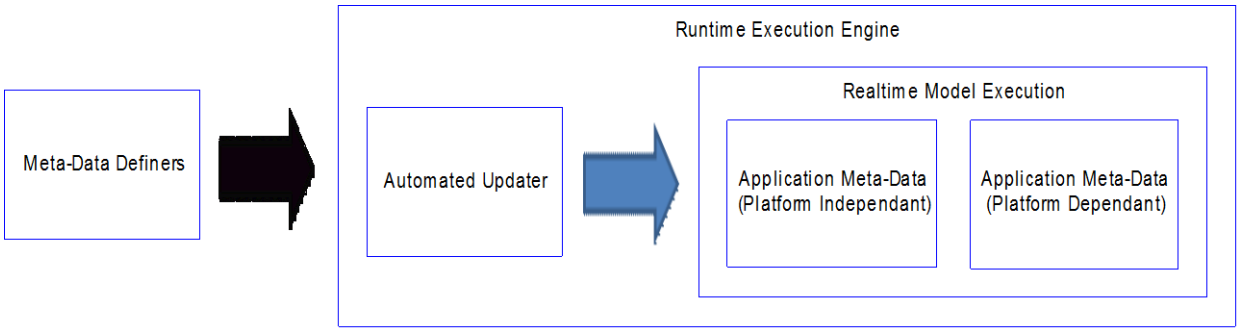


Figure 12 – Summary of the Temporal Meta-Model Framework

The meta-data model is platform independent and any supporting runtime execution engine must fully implement all features of the meta-model framework. Ideally, the runtime execution engine would be entirely platform independent however most realistic implementations will require some platform dependant components that are used to efficiently perform the low level processing and execution for their selected execution environment.

Figure 13 below presents a more detailed view of the temporal meta-data framework architecture for the development of meta-data EIS applications that; aids the specification of the target application, followed by the conversion of that specification into the appropriate meta-data that will then be interpreted by the framework's runtime execution components.

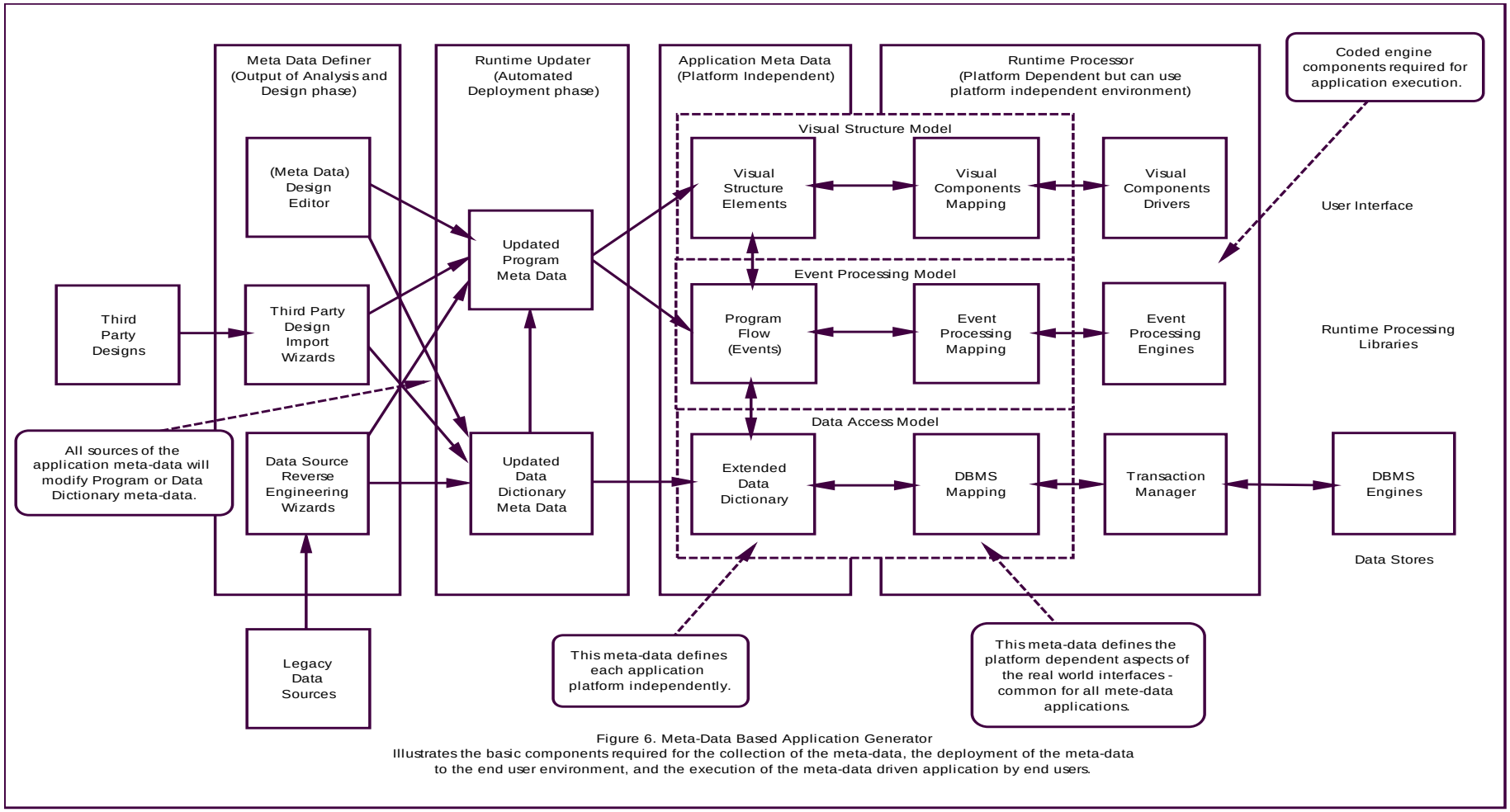


Figure 6. Meta-Data Based Application Generator
Illustrates the basic components required for the collection of the meta-data, the deployment of the meta-data to the end user environment, and the execution of the meta-data driven application by end users.

Figure 13 – Detailed view of the Temporal Meta-Model Framework

The general features of the temporal meta-data framework architecture for the development of meta-data EIS applications are provided in the following sections.

4.5 UML Notation for Temporal Meta-Model Framework

The conceptual modelling for the temporal meta-model framework has been developed using Sybase PowerDesigner [186]. This is a powerful enterprise CASE modelling toolset that supports the modelling of; business processes, requirements, data, enterprise architecture and UML application specifications. The toolset supports cross-model conversion and model maintenance, as well as template driven code and object generation.

While the ultimate output of the temporal meta-model framework is a user driven application generation framework, its design is a challenging technical problem which requires the modelling rigour of UML as an appropriate definition and capture mechanism. UML style notation overview diagrams are extracted from the full model throughout this thesis to illustrate the major points of the key design components. As the full model design report comes in at around 2000 pages it has been provided as a supplement to this thesis (see Appendix).

4.6 Detailed Representation of Temporal Meta-Data Framework Elements

The following section provides a more detailed explanation of the elements illustrated in Figure 13.

4.6.1 Meta-Data Definer

The meta-data definitions effectively become the application logic so it is crucial that efficient methods of defining the meta-data are available.

As the stored meta-data ultimately represents the output of the system specification process then minimal opportunities exist to expedite the original human business analysis effort although the ongoing potential for genuine simple application prototyping will exist for users of the meta-data EIS applications. However substantial shortcuts will be offered by reductions in the system design effort due to the amount of application infrastructure that would necessarily be provided by the framework's

runtime execution engine in support of the higher level components that the meta-data structure is modelled upon – hence limiting the design requirements to the business logic only, no need to specify the application infrastructure requirements which are provided by the framework. This automatic execution of the meta-data is a source of major savings in the development cycle.

4.6.1.1 Meta Data Design Editor

When starting a new meta-data EIS application design a comprehensive design editor is a requirement for the efficient specification and entry of the system design as stored in the meta-data. A custom editor is required that represents a suitable design paradigm for the meta-data EIS applications which is necessarily biased towards the production of the target meta-data syntax (see Chapter 7 - Accelerants for the Iterative Design of EIS Models).

4.6.1.2 Third Party Design Import Wizards

The field of system analysis and design has developed considerable expertise in providing Computer Aided Software Engineering toolsets to reduce the overall system development effort, particularly focussing on the introduction of UML based tools. Opportunities will become available to develop utilities that import design models from existing comprehensive third party design toolsets and convert these designs into the corresponding meta-data syntax.

4.6.1.3 Data Source Reverse Engineering Wizards

The database is an integral component of EIS applications which have a strong reliance on data dictionaries and rules based database transactions. Database schemas that do not obfuscate the original design by abstracting the names or types of schema elements, or by relocating database constraints or stored procedures to a remote system tier have significant potential as a starting point to the re-engineering of an existing system.

The development of utilities to reverse engineer existing database schemas and convert the schema to the corresponding data dictionary meta-data and thence to full supporting EIS application meta-data can accelerate the meta-data EIS application design process appreciably. Well-formed database schemas, without obfuscation, that seek to fully utilise the validation options of the modern Relational Database

Management Systems have the potential to reverse engineer directly to a fully working meta-data based application, requiring optimally minimal or no modification to the meta-data (see Chapter 7 - Accelerants for the Iterative Design of EIS Models).

4.6.2 Runtime Updater

The magnitude of the effort expended on future system maintenance for EIS applications represents by far the majority of the effort over the lifecycle of the EIS applications from the developer's perspective. Contributing causes to the ongoing high maintenance costs continue to be the;

- lack of consistent and available system documentation,
- inconsistently applied standards during different (re)development phases,
- lack of structured programming techniques,
- extension of the system to provide features that would have been better served by a full or partial redevelopment,
- natural attrition of software development team members knowledgeable of the system architecture,
- natural progression of the underlying technology to newer, richer and better supported platforms.

The indicated high level lifecycle maintenance figures estimates of 80% and higher refer to the costs for the developers of the system. For EIS applications this represents person years of effort for the developer. The use of meta-data based applications will drastically reduce these costs to the developer and user base.

For the many user organisations the greatly reduced automated update and deployment process of meta-data EIS applications will potentially reduce the regular update and upgrade periods from months down to days.

4.6.2.1 Updated Program Meta-Data

For the developer meta-data EIS applications will largely be self-documenting which reduces the risk and reliance on individual developers and development management practices.

For the user organisations, meta-data EIS applications are provided as streams of meta-data. Unless provided with an updated framework runtime engine (an expected occasional requirement), the same runtime engine as deployed to all users at a site would remain unchanged (although this is a relatively simple change management

scenario to resolve). Changes to a meta-data EIS application are simply a new stream of meta-data that represent only the specific application changes that are applied serially. To progress to any later version of a meta-data EIS application requires only the application of the correct sequence of progressive update version meta-data streams which will be executed and managed by the meta-data updater.

4.6.2.2 Updated Data Dictionary Meta-Data

Changes to the underlying data dictionary and associated database structures are invoked automatically as data definition commands embedded within the overall meta-data updates, which are interpreted and acted upon by the meta-data updater.

Any data locking and data migration requirements are managed automatically by the meta-data updater which can also allow the updates to be enacted on live systems if required.

4.6.3 Application Meta-Data

The ultimate aims of a meta-data model for EIS applications definition include abstraction from the physical constraints of any runtime components that are used for the final implementation and execution by the users as provided by the framework's runtime execution engine.

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

4.6.3.1 Visual Structure Elements

The visual structure meta-data is used to define the appearance of the meta-data EIS application as presented by the user interface runtime components to the users. Visual structure meta-data is analogous to the drag'n'drop forms functionality provided by modern GUI based Integrated Development Environment software and draws on advances such as XForms [187], [188].

4.6.3.2 Program Flow Elements

The program flow meta-data is used to define the user interface and local platform logical actions and procedures that are executed in response to user actions and other data changes. Program flow meta-data is analogous to the event processing

and general programming functionality provided by modern GUI based IDE and traditional programming languages.

A common paradigm in modern code generation that has become a virtual standard following on from the advent of event driven programming is a separation of the actions, implemented as events, from an underlying framework that provides a structure that makes logical sense for the individual applications.

The basic model depicted in Figure 14 provides for a separation of the structure and events, extending both to the managed hierarchies of the previous atomic structure and allowing for flow control between them. The model borrows heavily from the use of visual components to provide application structure - this is a commonplace analogy used in most modern IDEs for the code based generation of applications and from other research works in meta modelling [132], [189].

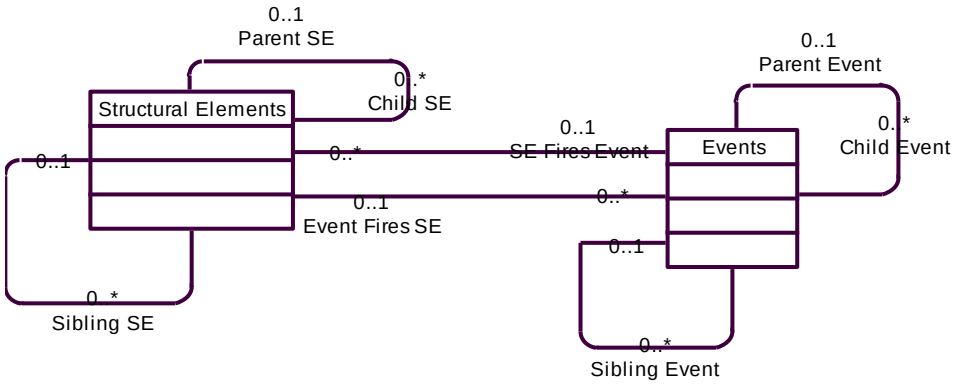


Figure 14 – UML based Notation Systems for Structural Element / Event Molecule

4.6.3.3 Extended Data Dictionary

The data dictionary meta-data is used to define the requirements of the database schema and the data changes, as transactions, required in response to user actions and other data changes. Data dictionary meta-data is basically analogous to the data dictionary role provided by modern RDBMS systems.

4.6.4 Runtime Processor

The meta-data must be interpreted and executed for the users. This role is supplemented by additional meta-data to map the generic application meta-data to the

interface requirements of the platform specific components for execution, plus the platform specific runtime engines that will perform the execution.

4.6.4.1 Visual Components Mapping and Drivers

Different user interfaces will consist of variations to the screen geometry, graphical ability, human interaction options and local functionality. Additional meta-data to describe the transformation of potential interface options is required to be defined as the intermediary layer between the core model meta-data and the end technology platform to process and deliver the user interface.

4.6.4.2 Event Processing Mapping and Engines

This layer optionally provides any transformation between the expected logical processing definition of the meta-data and any third party business logic, script processing and/or workflow engine. As the general logical processing requirements should be similar for all implementations, this meta-data would be expected to be primarily required for matching the meta-data logic to the fixed interface requirements of existing third party processing engines.

4.6.4.3 DBMS Mapping and Transaction Manager

There are a multitude of database systems or transaction processing engines that could be interfaced to the transaction meta-data. These transformations may be required to translate the atomic transaction statement components into a format more applicable to the interface of the database system or transaction processing engines. The base atomic structure of the transaction meta-data is designed to readily complement the standard SQL processing syntax.

4.7 Summary of Requirements of a Temporal Meta-Data Framework for EIS Applications

The definition of an EIS application in the context of this thesis was broadly defined in 4.2.4 Enterprise Information Systems Definition. This section further clarifies the core functional and technical features of EIS applications – features that must be satisfied by any EIS application development in the categories of:

1. **User Accessibility**: the features that a general EIS user has access to perform the business operations,
2. **Information Access**: the data access features that are provided by or for the EIS application to manage interactions with external data and systems,
3. **Systems Management**: features that the underlying EIS application is expected to support to assist ongoing management of the operation of the EIS application.

These features are further delineated by identifying how additional overall benefits are provided by a meta-data based approach vs the traditional hard-coded source code approach.

4.7.1 User Accessibility

These features need to be provided to support the daily operations of the general user interactions for users of the EIS application, falling into the often nebulous concept of "user friendliness". In general terms, the greater the features available in terms of ease of use, function and feature, the more effective the business users will perform in their use of the EIS application.

It is assumed that a modern EIS application will utilise the standard features of the common GUI whether deployed via thick or thin client technology.

4.7.1.1 Independent User Configuration

The ability for a user to modify aspects of an application's user interface to suit their individual preferences has been progressively improving although is generally limited to minor options such as; screen colours for the background, text and controls; fonts; some screen layout options; menu shortcuts or favourites.

As all application objects are defined in meta-data, the meta-data EIS application provides users with the ability to modify almost all static aspects of an applications parameters, not limited to just the basic user interface, but also to the entirety of the application layouts and general workflow of the intended application definition.

By static, as used above, I deliberately choose to limit the definition of the configuration to not altering any fundamental application intent or mandatory workflow logic. However, the meta-data EIS application can also offer dynamic modifications to these (as discussed in 4.8.2 Application Adaptability).

4.7.1.2 Configure Reporting

The majority of modern systems provide users with access to a view of the EIS data to facilitate custom reporting. Many systems will provide their own query or reporting tools, many will outsource this role to the use of third party reporting tools. The meta-data EIS application will also expose its internal data to third party reporting tools (as described in 4.7.2 Information Access).

There are many similarities to developing user interface screens and forms with developing report layouts. A report generator editor can be provided as part of the meta-data EIS meta-data editor, similar in function to the meta-data user screen designer (see Chapter 7 - Accelerants for the Iterative Design of EIS Models).

4.7.1.3 Multi-Lingual Application and Data

The vast majority of applications have been produced as single-lingual applications, with limited alternate linguistic versions optionally produced depending on the potential international popularity of the application. Such versions often tend to be based on the wholesale duplication of code with minor changes made to the new linguistic requirements to suit the changed textual aspects of the application. Later advancements in coding have provided improved support via external language interfaces to language text data files if the application is coded with the appropriate multi-lingual objects initially.

There are several aspects of multi-lingual applications to consider providing:

1. alternate language options, fonts and layouts for all application textual components including help, tutorial and lessons features,
2. for the appropriate orientation display of the alternate language elements,
3. for the appropriate orientation display of the user interface controls,
4. for the appropriate symbology and formatting of language specific data,
5. for any alternate or translated textual data

The first aspect above, alternate language options, are very simple to implement for a meta-data EIS as all text is already managed as data, thus the only allowance required is to allow for multiple language formats and associated user interface layouts. Further options for manual or automatic language translation can be considered; for manual, it is the responsibility of the definer to manage the appropriate translations for all text; for automatic, it is possible to refer to an online translation service to (at least initially) manage the translations.

The second aspect, language orientation, is managed by the framework runtime execution engine based on the best use of the available system interface orientations, the third aspect, for which in general the runtime engine would have minimal additional control over, and hence would generally not alter, as there may be limitations in the underlying operating system or GUI environments.

The fourth aspect, symbology and formatting, is generally already supported by computer operating systems or GUI environments.

The final aspect, translated data, is also very simply implemented for a meta-data EIS application. Similarly to the application text translation, both manual and automated translation options can be provided.

Additional options such as variable lingual audio clips for both application text and data can also be readily defined to facilitate features in the runtime engine such as audio on rollover. See 5.7.4.5 Multi-Lingual Entity Schema for Data for further detail of the model.

4.7.2 Information Access

These features are provided to support the more strategic requirements of the business by supporting interactions with external systems and data. Typically enacted by the business intelligence or information technology areas of the business, the external interactions, data and workflow will support a more integrated business environment.

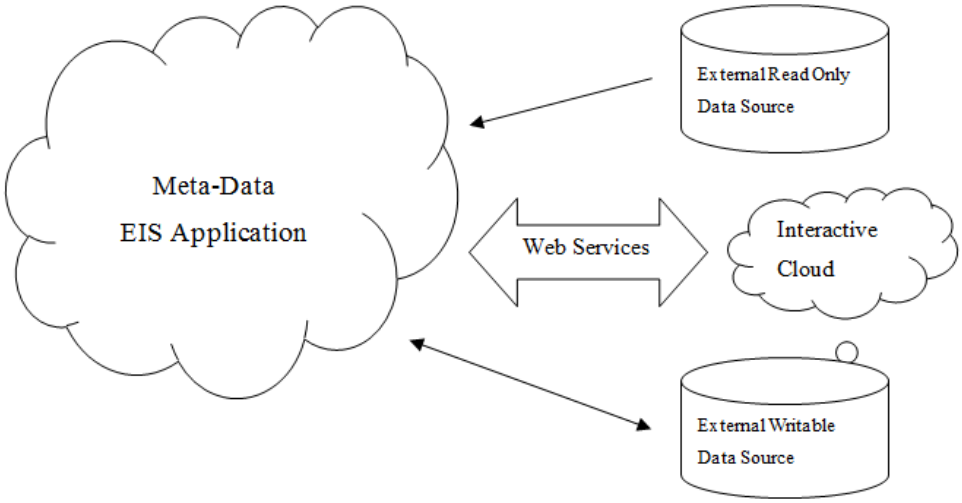


Figure 15 – External Interaction Overview

Figure 15 indicates the comparative interactions between common data based access to the meta-data EIS application vs the richer interactions available via web service interfaces directly to the framework.

4.7.2.1 Expose Internal Data

In a similar vein to accessing external data, the meta-data EIS must also be able to fully expose its internal data sources to allow external applications to access the data for additional processing as may be required. Due to the nature of the meta-data EIS application, this exposed data must include the EIS application data and may include the internal meta-data and any additional supporting data and meta-data.

At the most basic level, the external database used by the meta-data EIS application to manage its application data and meta-data is available for secure access to any permitted database user. However, there is typically a great difference between the raw data storage formats in a meta-data EIS application and the preferred data formats that database administrators and users like to work with. A meta-data EIS application will generally utilise entities and attribute names that are system generated due to the higher level of abstraction throughout a meta-data EIS application. While the abstractions are capable of modelling and mapping to user friendly acronyms by human operators, it is more efficient and less error prone to allow the meta-data EIS application to generate the more user friendly data abstractions.

Again, at the most basic level of the database repository, the meta-data EIS could generate user friendly database views and procedures that can be more readily used by external data operators to access and optionally manipulate the internal meta-data EIS application data (see Chapter 8 - Universal Access to Temporal Meta-Data Framework for EIS in the Cloud).

Similarly, more advanced functionality such as providing access to meta-data EIS application functionality to external users can be provided by the auto-generation of, e.g. web service commands, that can more securely permit access to both the meta-data EIS application data and the defined functionality of any and all aspects of the meta-data defined application features.

The meta-data approach is thus able to provide external users with complete and secure access to not only the meta-data EIS application data, but also meta-data EIS application functionality. Additional access to the internal meta-data data and meta-data processing commands can also be exposed for external users.

4.7.2.2 Access External Data

An EIS cannot be an island with regard to data – it must be able to access data from external sources to integrate with the internal EIS data to minimise organisational duplication of data and provide additional data collation and processing functions.

The majority of EIS applications have some capability to interact with external data sources ranging from simple read only data access for report inclusion, through to the posting of database transaction results to external data sources. The inherent disadvantage of traditional EIS applications is that their level of data access is limited by the specific coded functionality that has been established for that system.

A meta-data EIS application necessarily requires advanced internal data access and processing capability for all of its application functionality, the definition of which is itself stored as data as meta-data. The ability to interact with external data sources becomes virtually unsurpassed as the meta-data EIS application is able to potentially treat the external data with the full functionality that is available to the meta-data defined internal data stores of the meta-data EIS application.

The only limitation to the level of direct interaction by the EIS with the external data stores are any security policies defined by the owning organisation of each external data store. As the meta-data EIS application typically requires additional supporting data entities and attributes that optimise the meta-data EIS operations for each data source, whether internal or external, the EIS may not have appropriate control over the external data source to enact these modifications. However, the meta-data EIS application can manage the additional optimising supporting entities and attributes on a purely internal basis supplemented by transparent mapping to the external data rows, to effectively manage the external data source transactions and incorporate the external data source into the meta-data EIS application data framework, thus providing the optional full functionality with the external data sources.

The meta-data approach is thus able to provide the fullest data interaction capability with any external data set as long as the external data can be accessed by the implemented database engine.

4.7.2.3 Expose API or Functions

In a similar vein to exposing internal data, the meta-data EIS application can also fully expose its application functions to allow external applications to securely execute aspects of the meta-data EIS application from other external applications, whether locally or from in the cloud.

As every function of the meta-data EIS application is modelled to an atomic level, and internally executed at that level, similarly, that execution can be invoked by an appropriate authorised external command – the syntax of these commands, e.g. web service commands, are automatically generated based on the meta-data definitions of the application functionality and thus can provide equivalent and secure external invocation of transactions with identical security and data integrity as though executed within the meta-data EIS application.

The use of standards such as web services to provide this functionality readily opens up the functionality of the meta-data EIS to almost any modern information system or web page, providing the foundations of not only ubiquitous data access but also ubiquitous data processing.

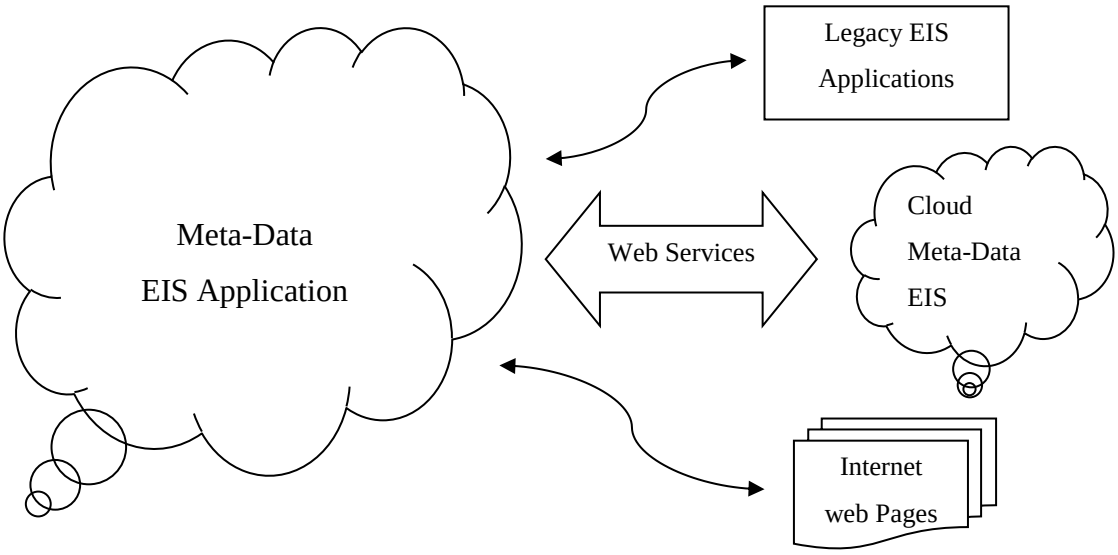


Figure 16 – External Application Calls to the Framework –

Figure 16 indicates the indicative interactions between external application options and the meta-data EIS application via the framework. Rich interactivity is possible from any external system depending on its level of programmed interaction

although richer interactions are likely between multiple meta-data framework based applications due to their identical interface.

4.7.3 Systems Management

These features need to be provided to aid the application and systems administrators in the daily and ongoing management of the EIS to assist with ensuring speedy access to and maximum uptime of the applications to the business.

4.7.3.1 User and Group Security

Access security is a fundamental requirement to ensure that only appropriately authorised transactions are enacted in the EIS. Once users have been assigned to security groups there is a similarity between the requirements and implementations of User and Group Security as per the discussions for 4.7.3.3 Audit Logging, to provide the mechanism to permit user interactions in the EIS application. The operations include key activities such as user login, module or function access, data access and data actions.

Traditional systems development would define systems users as one of the following primary methods:

- Pre-defined users and/or groups,
- Individually define users (with no groups),
- Individually define groups (with no users) - rarer,
- Define both users and groups – can be optionally further subdivided with options such as; are individual users defined or treated differently than groups; can users be in multiple groups; can groups be defined within groups. Each sub-option supporting alternate security functionality.

Clearly the latter option provides the most flexibility and regardless of the desired sub-option, is capable of readily identifying a user to the meta-data EIS application. Additional synchronisation options can include the ability to interact with various operating system and/or database level directory or user authorisation services, that can reduce the duplication of user account information on separate application systems such as the meta-data EIS application.

Traditional systems development would provide a User and Group Security capability using one of the following primary methods:

- Hard-code into the application each specifically defined execution condition, action and invoking requirement,
- Hard-code into the application the availability of a common user authorisation procedure and invoke based on user access tables,
- Hard-code into the logic processing, database system or transaction processing engine using one or both of the above styles.

Arguably the flexibility increases with each of the above options as you progress the list, while the implementation effort decreases. Alternatively, the meta-data approach is implemented as follows, again similar to audit logging functionality:

- As every function of the EIS application is modelled to an atomic level, and executed at that level, the authorisation level can also be triggered at any hierarchical level from the root node of the model down to the atomic function level.
- A single user authorisation processing service is coded once into the runtime engine which is then re-used for every new meta-data EIS application that is modelled and executed i.e. no subsequent effort by any EIS system modeller or end user following deployment.

The meta-data approach is again clearly superior as is provides the ultimate flexibility to define the authorisation requirements in any combination of user and/or group, and function, and is entirely configurable. See 5.6 Secure Access and Authorisation for details of the model.

4.7.3.2 Data Archiving

The concept of data archiving has many definitions and instantiations including options such as:

- Providing a data backup feature for data security,
- Providing a full database archive capable of supporting application execution, i.e. snapshot,
- Marking deleted records as deleted vs deleting deleted records from the database,
- Duplicating changed records with timestamps to record data changes vs overwriting data.

As the current state of the data in any database only reflects an abstraction of the data at that single point in time, data archival options are an attempt to record elements of the history of data transactions that have led to the current state. At the lowest end of capability are application or database systems that record no temporal transaction information (when the transaction occurred) and always overwrite the latest data. At the highest end of capability are fully temporal applications or database systems that store enough transaction information that will allow the reconstruction of the state of the database to any point in time (see 4.8.1 Temporal Execution).

It will be shown further in this thesis how the meta-data EIS application can readily support fully temporal applications with even greater functionality. A key aspect of supporting this functionality is also the provision of full data archival functions, i.e. The storage and maintenance of all data required to retrieve and display the state of the database at any point in time.

Full data archival functions are not necessarily radically easier to implement in the meta-data EIS application vs the well-designed traditional EIS although the necessarily central meta-data processing engine that manages all transaction processes does simplify the overall effort by coding once and thus achieving its functional reuse for all subsequent data transactions.

4.7.3.3 Audit Logging

Audit Logging provides a record of user interactions that have occurred in the EIS application. This should include key actions such as user login activity, module or function access, data access and data actions.

Traditional systems development would provide an audit logging capability using one of the following primary methods:

- Hard-code into the application each specifically defined audit condition, action and invoking requirement,
- Hard-code into the application the availability of a common audit procedure and invoke audit procedures based on user access and audit tables,
- Hard-code into the logic processing, database system or transaction processing engine using one or both of the above styles.

Arguably the flexibility increases with each of the above options as you progress the list, while the implementation effort decreases. Alternatively, the meta-data approach is implemented as follows:

- As every function of the meta-data EIS application is modelled to an atomic level, and executed at that level, the audit level can also be triggered at any hierarchical level from the root node of the model down to the atomic function level.
- A single audit processing service is coded once into the runtime engine which is then re-used for every new meta-data EIS application that is modelled and executed i.e. no subsequent effort by any meta-data EIS application modeller or end user following deployment.

The meta-data approach is clearly superior as is provides the ultimate flexibility to define the audit requirements in any combination of user and function, and following first time deployment is entirely configurable. 5.2.1 Generic Distributed Temporal Meta-Data Inheritance describes the tracking model components.

4.7.3.4 Encrypt Individual Data

Access to EIS data is generally managed by the authorisation procedures inherent to the EIS and/or database system in use. Data is typically only further encrypted as required by any higher security provisions of the application.

To provide EIS-wide encryption of data requires the EIS to manage the security and storage of the keys and the algorithms employed in the encryption and decryption.

Individual or group user data can also optionally be provided with storage encryption however the simplest method is to use the similar EIS encryption algorithms but allow the users to specify their own keys. The operational use of this optional feature is dependent on the security policies of the organisation as key management becomes the responsibility of the user and a lost key would likely result in loss of access to the data. Accordingly, this feature would be recommended for use only by non-core enhancements to the meta-data EIS.

4.8 Summary of Enhanced Features Provided by the Temporal Meta-Data Framework

The temporal meta-data framework doesn't just enhance the development and delivery of traditional EIS functions, over that developed by traditional means. The temporal meta-data framework also provides substantial new functionality as well as greatly simplifying the deployment of advanced features that require significant effort when delivered by traditional systems development.

This section describes these advanced features that can be readily provided by the temporal meta-data framework in its delivery of EIS applications, in the categories of:

1. **Temporal Execution**: features that support a varying historical or temporal basis for the meta-data EIS for both data and application logic,
2. **Application Adaptability**: the ability for even non-technical users to dynamically modify the structure, logic and workflow of the meta-data EIS application – without coding,
3. **EIS Application Deployment**: features that facilitate rapid, even immediate, deployment of the changing meta-data EIS application into operational production use,
4. **User Knowledge and Education**: the provision of relevant and accurate education materials to the technical and business user base in online and offline environments.

It is further clarified where these features are unique to the temporal meta-data framework or where the temporal meta-data framework provides substantial optimisation and efficiency vs the traditional hard-coded source code approach.

4.8.1 Temporal Execution

Temporal data management is a well understood field as it applies to the common database. However, the temporal meta-data framework can allow meta-data EIS applications to also execute across time, regardless of the meta-data EIS application version changes that have occurred, providing a unique capability.

4.8.1.1 Temporal Data Management

Whilst temporal data management is a well understood field, embedded temporal solutions have never become standardised in the major SQL database vendors.

Individualised solutions offering varying levels of complexity and functionality are relatively straightforward to implement however they are often hampered by the associated problem of a non-temporal execution base i.e. minor and major application version changes may regularly limit the availability of a useful temporal window for that incremental application version. This is a major reason why widespread adoption of temporal data management has not been implemented.

The basic requirements for temporal data management are:

- Data is never deleted – it is only marked as deleted at the appropriate time and date.
- All changes to data are recorded and time stamped.
- The temporal structure is organised to facilitate simple data management and access to current and historical data.

The time period for which such temporal data access is maintained and available can be defined as the Temporal Data Window.

There are a variety of options for the structure of data in an implementation of temporal data management:

- All temporal data managed in the same production table – complex additional temporal management structures, complex data queries, potential performance and archival issues,
- Current data managed separately with all historic data managed as external copies of the production tables updated in real-time – simplifies the access to and performance of current data for most operations, temporal management and access can be more easily offloaded for adhoc access.

Temporal data management needs to be "under the hood" i.e. it should be the function of the core transaction engine and while optimised within the meta-data EIS application transaction engine, the development effort is of a similar magnitude to traditional development. However there are unique benefits to the meta-data EIS that are direct enablers of the more substantial feature provided by the temporal meta-data framework, as discussed in the next section. See 5.2.2 Generic Distributed Temporal Data Inheritance for the model components.

4.8.1.2 Temporal Meta-Data Management

The meta-data in a meta-data EIS application serves as the application definition, analogous to the source code for traditional application development. Source code is compiled into executable modules and while both are subject to version control it is very uncommon for EIS applications to be dynamically managed such that multiple versions of its application modules are dynamically executed based on a defined conglomeration of these modules into specific overall application versions.

A unique feature of the application of temporal data management techniques to the atomic meta-data elements of the meta-data EIS application can 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. The time period for which such temporal application access is maintained and available can be defined as the Temporal Application Window.

This unique solution provides many useful operational benefits to EIS user organisations that can provide significant savings in effort, time and resources:

- Maximises the currently available temporal application window – depending on the frequency of patches and updates, and the structural changes imposed, this can increase the temporal window from periods of days or weeks, up to the entire operational life of the meta-data EIS application i.e. the full operational lifespan of temporal execution is available, always.
- No longer require the resources used to enact and execute any internal data archival or rollup, or as application functionality is changed due to typically irreversible EIS version changes or upgrades.
- Eliminates the time required for making the temporal access available, while enacting the above processes, as well as eliminating any operational downtime that may be required – in general, the access to any temporal application window should be immediate.

The Temporal Application Effectiveness of an overall application environment lifecycle can be defined as the multiplication of the Temporal Data Window and Temporal Application Window to provide an indication of the maximum temporal

accessibility of the system architecture as a whole. A maximum score would require infinitely continuous periods for both parameters which is not practical, however in practice could be achieved by systems demonstrating an ongoing continuity of an effective system architecture such as exhibited by model evolution of the meta-data EIS application and its framework runtime engines.

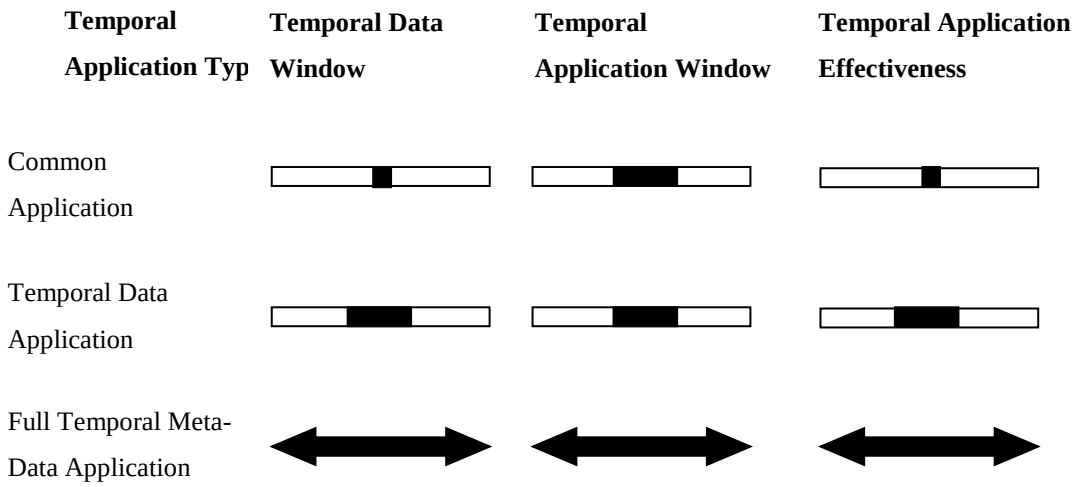


Figure 17 – Comparison of Temporal Application Effectiveness

Figure 17 above provides a comparison of the relative magnitudes of the available temporal application windows of traditional vs meta-data EIS applications where:

- **Common Application**: an application (whether presented as a thick or thin client, and executed from a static non-temporally varying codebase) accessing a database schema without temporal data management features.
- **Temporal Data Application**: an application (similar to the Common Application) accessing a database schema with effective temporal data management features.
- **Full Temporal Meta-Data Application**: an application (whether presented as a thick or thin client, and executed from a dynamic and temporally varying codebase) accessing a database schema with effective temporal data management features. E.g. meta-data EIS application in this thesis.

This capability can also be provided by traditional development methodologies which employ a dynamic version management code execution model, and maintain temporal data management for the EIS application, as an effective temporal execution solution. However when also combined with other unique benefits of the meta-data EIS application (see 4.8.2 Application Adaptability), which drastically increase the effective range of user generated and user controlled versions, the ability of a traditionally developed EIS application to manage temporal execution effectively diminishes to zero. See 5.2.1 Generic Distributed Temporal Meta-Data Inheritance for the model components.

4.8.1.3 Temporal Rollback and Rollforward

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.

As all transaction executions are recorded (as described in 4.7.3.3 Audit Logging) and the subsequent results of changes to the data base are recorded (as described in 4.8.1.1 Temporal Data Management) and while any changes to the meta-data EIS application are tracked (as described in 4.8.1.2 Temporal Meta-Data Management) then 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 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.

4.8.2 Application Adaptability

EIS applications consist of three common layers or at least conceptual considerations for development; user interface, business logic and the database repository. Traditional EIS application development almost exclusively requires highly trained developers fluent in the various and often multiple languages, protocols and technologies that constitute the EIS application.

A major initiative for the meta-data EIS application is that generally only higher level models and abstractions need to be defined by the business users, with some lower level, and the temporal meta-data framework runtime engine will then provide all execution of these models based on the already established execution code developed for the runtime engine.

A further significant innovation of the meta-data EIS application is that the meta-data models, which act as the source of the meta-data EIS functionality, can be readily modified by non-technical business users, including non-core or non-global functionality that can be for private or functional group use – all without coding, and with generally rapid or immediate deployment for operational use by the organisation. 5.4.5 Variant Logic describes the model for this adaptation.

4.8.2.1 Independent Dynamic User Data Store Configuration

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.

Indeed, the highest level of approval can also alter the definition of the core meta-data EIS application data stores, which is analogous to acting as the "system's developer", and is an expected role during the EIS application meta-data "definition" or "development" – this is a subset of the EIS application meta-data definition role (see 4.8.2.4 Modify Core and Non-Core Application Functionality).

Thus the meta-data EIS application readily permits access and interaction to and with existing or new data stores, at either the personal user level (single or permitted group access) or on the global level (becomes part of the core meta-data EIS application definition). All of the necessarily extensive data access features of the meta-data EIS application as provided by the runtime engine become available for use for the additionally defined data store management.

4.8.2.2 Independent Dynamic User Interface Configuration

Similarly to the previous section, the definitions for user interfaces and logical workflow in the meta-data EIS application are defined in data, hence the authorised user is also able to optionally modify and define additional application features to operate with, enhance or optionally replace existing application functionality – without coding, and for immediate execution.

Again, the highest level of authorisation can also alter the definition of the core meta-data EIS application features, again analogous to acting as the "system's developer" (see 4.8.2.4 Modify Core and Non-Core Application Functionality).

The meta-data EIS application readily permits the full range of application feature set changes to interaction with existing or new data stores, user interfaces and logical workflows - without limitation other than that imposed by logical integrity and authorisation.

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 highest level designers. Within this scope users can, within their authorisation limits;

- Remove (not delete) non-mandatory features – e.g. Remove an entry field that is not used by a particular user or role on a particular screen,
- Relocate any features between user interface locations – e.g. Re-arrange a user interface screen or re-arrange objects between multiple screens,
- Modify non-mandatory features – e.g. Change the text for a screen object to be more specific to that user, or re-define a text entry field to a drop-down selection where there may typically be only a limited choice for that user,
- Define new features to support new or existing data stores – e.g. Define the user interface entry requirements for newly defined personal data storage.

At the global level, becoming part of the core meta-data EIS application definition, changes can be made to the application meta-data as a whole affecting all users, otherwise access is as granted by the owning definer.

4.8.2.3 Independent Dynamic User Logic and Processing Configuration

Continuing with the theme of user meta-data modification, the definitions for data manipulation and processing in the meta-data EIS application are also defined in meta-data, and also able to be optionally modified and provided with additional application features by users. The highest level of authorisation can also alter the definition of the core meta-data EIS application features, as the "system's developer" (see 4.8.2.4 Modify Core and Non-Core Application Functionality).

Data manipulation 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.

These functions arguably represent the most complex technical knowledge that a competent user would need in order to change existing or define new EIS application functionality within the meta-data EIS application. Given the widespread usage and adoption of tools such as spreadsheets throughout the business world, this level of skillet is considered to be a reasonable acceptable level for the meta-data EIS application meta-data definer that still provides an order of magnitude of accessibility over mastering the alternate skill sets required as a traditional EIS code developer.

4.8.2.4 Modify Core and Non-Core Application Functionality

In a traditional software development the source code that is compiled to produce the application does not usually have any particular significance in terms of its level of security or role in the application – rather it is how the software is designed to work and the appropriate runtime authorisations that may then set such roles. In terms of modularity an increasing number of software systems will allow the development of third party plug-in modules that can provide user defined functionality to interoperate

with the core software system – these modules tend to be themselves examples of additional traditional software development to provide the functionality.

In a meta-data EIS application there are various levels of functional authorisation that can be defined to support the range of a multitude of application logic owners such as; core application vendor, regional application maintainer, site owner, group and user maintainers. Thus a flexible hierarchy of functional ownership can be defined within the meta-data and managed via the standard application integrity processes of the meta-data definitions. These functional authorisation processes are governed by the following:

- All original meta-data is owned by the identified core supplier (whether internally defined or externally procured) and represents the highest level of authorisation for that meta-data EIS application.
- Additional functional owners can be defined as lower level authorisations which can include additional external vendors, corporate or local users.
- Meta-data owned by one functional owner cannot be modified by a different functional owner, to ensure application semantic integrity.
- Any functional owner can define new meta-data, reference and invoke meta-data owned by other functional owners (where authorised), and modify undefined meta-data attributes of meta-data that owned by other functional owners where this functionality has not been restricted by the functional owners.
- Meta-data defined by a higher level functional owner always over-rides any other identical meta-data definition created by a lower-level functional owner – although this may on occasion identify changes that may then be required to be made by lower-level functional owners.

Hence, the core meta-data can be extended by any combination of authorised users to provide enhanced functionality in the meta-data EIS, while also maintaining application integrity, and while maintaining the ability for the core and other higher-level functional owners to securely provide valid updates to their own meta-data definitions.

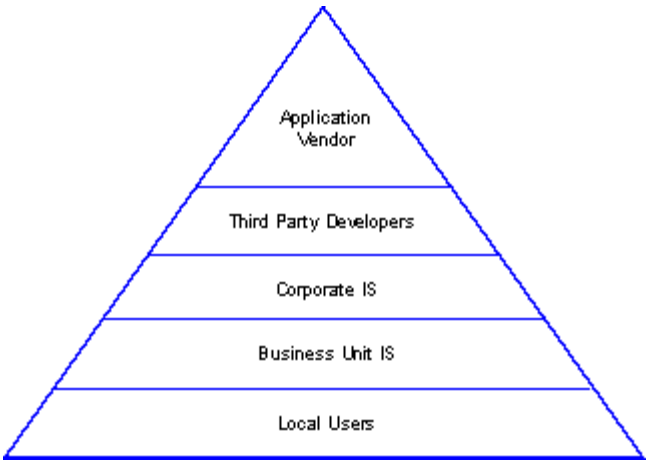


Figure 18 – Example hierarchy of Meta-Data EIS Application Extensions

Figure 18 above demonstrates a potential hierarchy of defined meta-data EIS application definers, from the highest level original vendor or supplier down through to the meta-data definition changes available to the end users, although not limited by functionality regardless of their position in the authorisation hierarchy.

This dynamic editing feature of the meta-data EIS is drastically different from the traditional development lifecycle, providing genuine real-time and distributed rapid application development capability, and greatly reducing the incidence of locked down or closed EIS application eco-systems.

4.8.3 EIS Application Deployment

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 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, plus the current reality of real world EIS implementations that typically require several months to implement new or upgraded versions.

The meta-data EIS application can drastically reduce these delays due to the wholesale change in the development methodology lifecycle (see 4.9 Temporal Meta-Data Methodology for the Meta-Data Based Application System Lifecycle) and the

unique meta-data update deployment model, which can reduce the overall deployment delays down to days or even virtually instantaneous distribution and update.

4.8.3.1 Automatic Application Meta-Data Version Control

Traditional well managed software development will utilise version control systems which will manage the tracking and identification of all source code changes that occur in a particular development phase, whether as a minor patch or major upgrade. The difference between entire application or EIS versions may generally represent thousands of lines of code changes.

In the meta-data EIS application, similar functionality changes may require just a few, or perhaps hundreds of individual meta-data changes, each change occurring one at a time during the meta-data editing process, or can occur simultaneously by multiple meta-data definers when multiple modifications are being managed. Hence the overall meta-data application integrity is maintained by executing each individual change serially – in fact this is also how meta-data EIS instances are updated with new versions (as a stream of new changes, as discussed in the next section).

As each meta-data change is separately identified and logged, they can also be categorised for batch identification, and the series of relevant meta-data changes can be clearly identified as constituents of an incremental release. When a release is ready, the series of meta-data changes is extracted to be executed in sequence on the destination meta-data EIS applications as the new update.

4.8.3.2 Immediate Deployment

In traditional software development the standard lifecycle requires a period of test and verification prior to general release. The update may be released as an entire package or incremental subset of components that need to be distributed to user organisations for their own testing and internal deployment. EIS updates can often require considerable efforts by skilled resources to implement and migrate to, often requiring months, and the complexity of EIS systems and their high importance to an organisation will usually dictate an extensive internal test and verification program.

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 never 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.

The meta-data EIS application provides a drastic simplification of the update process. As discussed in the previous section, the meta-data EIS is updated by a sequence of meta-data changes implemented in series. Often these meta-data changes will have minimal effect on a live system, although changes that affect existing data will require the execution of the appropriate system generated data modification to occur, which may require temporary locking of or restricted access to that feature by the runtime engine which will be managed automatically.

It 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 would almost always provide great optimisations and significant savings in time and resources.

As mentioned in 4.8.2.4 Modify Core and Non-Core Application Functionality any authorised meta-data update may over-ride other identical meta-data functionality defined by other lower-level functional owners. The meta-data update process can identify these occurrences during the update and prepare a report of recommended 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 (see 4.8.4 User Knowledge and Education) to assist users with the exact nature of the transition.

4.8.3.3 Merging Multiple Meta-Data EIS Applications

The issue of merging source code based applications is very problematic, particularly when involving code from disparate sources, due to the typical unsuitability of available source code for software merging. The issues of different languages, coding standards and styles, designs and templates, all contribute to an almost unsolvable problem – often overcome only by implementing an expensive redevelopment solution. The meta-data EIS application simplifies this process as the meta-data models are structurally identical which removes the greatest technical compatibility challenges.

To complete a merging of the meta-data EIS applications still requires the remaining semantic application issues to be identified. Disparate application logic is relatively simple as the associated meta-data can be largely used as is. More analyst interaction is required when there are functional similarities to application elements as these semantic unions need to be resolved. I have identified several logical tools to assist with meta-data EIS application merging:

- **Standard Element Referencing**: the simplest merge option involves creating new references in one model to existing elements in a second model to most readily provide access to application features of the second model to users of the first model.
- **Virtual Data Element Mapping**: provides infra-structure level merging and integration of similar data type elements between multiple models that achieves an underlying rationalisation of relational data structures.
- **Element Envelopment**: the highest level of model integration is to absorb an element from one model as a virtual instantiation of a similar element from the other model.

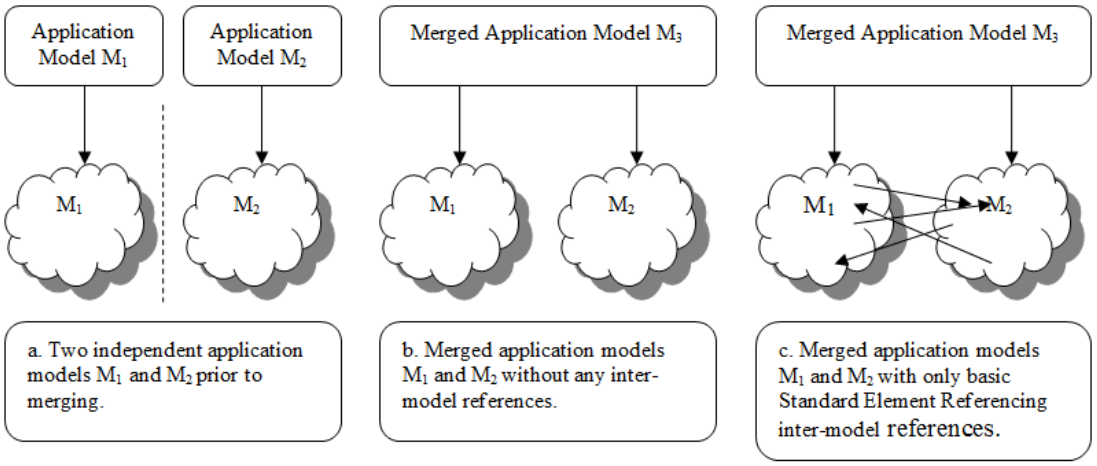


Figure 19 – Simple Standard Element Referencing Merging

Figure 19 demonstrates the initial steps in merging separate application models into a new model, creating basic inter-model integration between the models using the simpler only Standard Element Referencing inter-model references.

By removing the technical source compatibility and simplifying the meta-data merging, the meta-data EIS can provide a greatly simplified development process to merge applications.

4.8.4 User Knowledge and Education

The oft quoted bane of a technical developer's life is to prepare documentation, whether inline to the source code or the design documentation, or externally as user documentation – while certainly not a universal truth, this common urban myth has been in existence for as long as computer programming has been.

The meta-data EIS can resolve these potential problems as the structure of the meta-data in combination with appropriate extraction templates can automatically generate documentation in the classically user friendly formats for the provision of relevant and accurate education materials to both technical specialists and the business user base in online and offline environments See 5.2.1 Generic Distributed Temporal Meta-Data Inheritance for the supporting model components.

4.8.4.1 Automated System Design Documentation Generation

Traditional software applications generally require any additional descriptive system documentation to be explicitly developed as separate activities. Such documentation may include; system design, test plans, deployment plans, user

manuals, help files etc. For the most part, these documents will be required to be manually generated although advanced developers using Computer Aided Software Engineering tools may receive some accelerants where the CASE models can be used to generate some level of documentation.

To an extent, the meta-data editor component of a meta-data EIS application is a very advanced CASE tool and a reporting component can be used to assist in the generation of a variety of useful documentation that will aid the meta-data EIS application's human users in the understanding of the purpose and structure of the meta-data EIS application.

As the basic EIS meta-data is structured for the generation and execution of meta-data EIS applications, extracting ranges of meta-data for alternate reporting purposes such as for various system design documents can be readily provided. The typical mechanisms for extracting the meta-data into the reports will be via templates for the required document types or direct extraction from the meta-data structure via external data access (see 4.7.2.1 Expose Internal Data).

Another major benefit of the meta-data EIS application is that user interface graphics can also be auto generated by the frameworks runtime execution engine directly from the meta-data and incorporated into the output document along with the structured text, rather than requiring the laborious task of manually creating screenshots, then editing and pasting into each document as required.

4.8.4.2 Automated User Assistance Documentation Generation

The basic EIS meta-data structure will provide all of the required structured meta-data to support and explain the technical structure of the meta-data EIS application to technical business analysts and information technology specialists, especially those involved in the integration of the meta-data EIS to legacy systems. Design extraction reports can be provided as described in the previous section.

However, the issues of providing more user friendly information to typical EIS application users requires a more delicate touch generally requiring more verbose and descriptive language than might be developed for the technical specialist. While the basic EIS meta-data is structured for the generation and execution of meta-data EIS applications, additional descriptive meta-data can be included to provide this additional textual and descriptive information that can supplement the basic EIS meta-

data to enhance the overall user friendly quality of any collated document output that may be used for generating user manuals or online help.

The mechanisms for generating the user assistance documentation are identical to those for the technical documentation as discussed in the previous section, but in this case will require an additional level of effort to provide any additional textual and descriptive information that will be more readily addressed to aiding the understanding of the meta-data EIS application for the typical business user.

The immediacy of the meta-data EIS application always provides an up to date version of any documentation which is generated from the same EIS meta-data that generates the actual meta-data EIS executable application, thus removing any possibility of incorrect synchronisation of user documentation with EIS application version, which is a common occurrence in traditionally developed systems due to the lags often introduced in finalising the documentation.

4.9 Temporal Meta-Data Methodology for the Meta-Data Based Application System Lifecycle

Recent technology advances have not fundamentally altered the outcomes of the common system development methodologies used to develop Enterprise Information Systems (EIS). Variations of these Waterfall, Spiral, Fountain and V models are still maintaining the basic paradigm for system development as the traditional stages of analysis, design, code, test and deploy (see Figure 20). New system development methodologies such as Prototyping and Agile Processes can provide specific advantages but they are not guaranteed to change the overall magnitude of the total development effort.

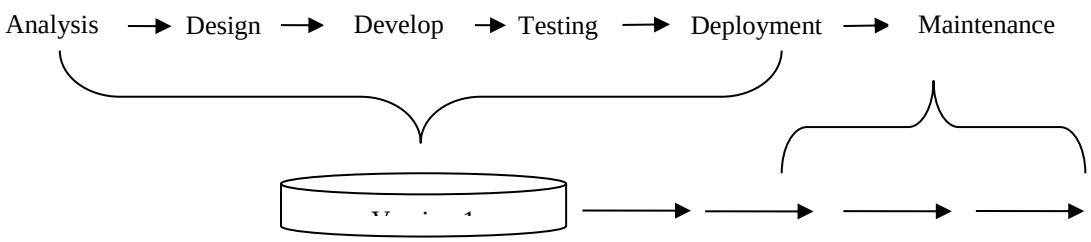


Figure 20 – Standard Development Methodology

This thesis asserts that performance of the requirements analysis and efficient collection of this information can also perform the bulk of the design phase for a meta-data EIS application, largely as a simultaneous activity. With the collective design requirements recorded as a model in the described meta-data structures, the meta-data EIS applications will be executed automatically from the meta-data model by the runtime engine of the temporal meta-data framework.

Expected major savings in time, resources and effort will be further recognised by the virtual removal of the development, test and deployment phases which commonly account for over 50% of the current system development effort. It would generally not be prudent to fully ignore testing, however it is expected that significant effort savings would be provided by the meta-data EIS application due to; automatic generation of test plans, reduction in testing efforts focussed on only the required application logic rather than all aspects of the runtime engine for the temporal meta-data framework which can be considered as common and pre-tested. Deployment is also greatly minimised as discussed in 4.8.3 EIS Application Deployment.

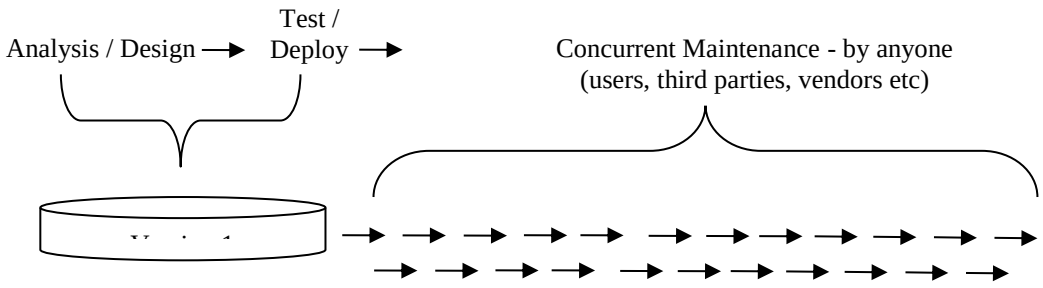


Figure 21 – Temporal Meta-Data EIS Application Methodology –

Figure 21 illustrates the shorter development lifecycle of the meta-data EIS application in addition to the faster and more frequent customisation turnaround and upgrade lifecycles.

Meta-data EIS applications are also largely self-documenting, based on the EIS meta-data (see 4.8.4 User Knowledge and Education), which reduces the risk and reliance on individual developers and development management practices and also reduces the development time and effort.

Overall substantial savings, conservatively estimated at well over 50% for the initial development or deployment are expected to be obtained by the employment of the temporal meta-data framework for meta-data EIS applications. Yet these savings only applies to the first iteration of each new meta-data EIS application that is developed, with even greater ongoing costs savings to be expected throughout the lifecycle.

	Estimated % of Effort in Projects (A)	Initial Development / Deployment Savings (B)	Per Phase Savings (A*B)	Ongoing Maintenance Savings Per Cycle (C)	Per Phase Savings (A*C)
Analysis	6%	0%	0%	30%	2%
Design	13%	50%	7%	75%	10%
Develop	40%	95%	38%	95%	38%
Testing	21%	80%	17%	80%	17%
Deployment	20%	0%	0%	95%	19%
	100%	**Initial Saving:**	**61%**	**Ongoing Savings:**	**85%**

Table 2 - Estimated Initial and Ongoing Maintenance Savings

Table 2 illustrates the high level of potential savings that can be achieved by developing and implementing a meta-data EIS application compared to a standard EIS. Higher levels of savings are expected to be achieved for each ongoing maintenance release as indicated. This analysis does not include the additional benefits of ongoing user based or internal customisations, known as Variant Logic, to the meta-data EIS application which depending on their magnitude of use could be expected to provide further significant ongoing savings.

The ongoing maintenance and upgrade of the meta-data EIS application can be expected to obtain even higher levels of ongoing optimisation and savings due to;

- Field enhancements to the meta-data EIS applications as developed by users can be reviewed as part of a new genuine business partnership process between the original meta-data EIS vendors and users. These then already existing user validated improvements can provide a directly available core basis of meta-data design to build upon and/or incorporate into the standard meta-data EIS application functionality of future versions, accelerating vendor innovation.

- The meta-data EIS application is always fully documented which minimises the risk of key design knowledge not being transferred, and maximises the education and knowledge transfer between new developers (meta-data definers).
- User testing and verification can be minimised as only the changes to the meta-data EIS application, as clearly identified by the meta-data update process (see 4.8.3.1 Automatic Application Meta-Data Version Control) need to be tested. Similarly, user education can be streamlined as only the clearly identified changes of functionality as available to the defined system roles of users need be presented as the training update.
- The meta-data updates are self-updating, simplifying the setup and maintenance of any test system environments i.e. at worst another test instance of the meta-data EIS application is created for testing rather than any potentially complex specific re-installation of a legacy EIS system.
- Larger user organisations need only maintain local meta-data EIS specialists rather than a local team of developers to assist with any local needs for meta-data customisation.

A major issue of current EIS system updates is that they can often take months of effort becoming major organisation projects – with a meta-data EIS this can be reduced to days or even hours.

Upgrade Scope	% of Effort vs Original Deployment (A)	Number of During EIS Generation Lifespan (B)	Sum Upgrades for Standard EIS Cost 1+(A*B)	Relative Sum Upgrades of Meta-Data EIS Cost (see earlier)
Minor	15%	10	150%	22%
Major	40%	5	200%	29%
		Total Generation	**350%**	**51%**

Table 3 - Estimated Relative Generation Costs

Table 3 illustrates the relatively lower costs incurred by the meta-data EIS application for all through-life upgrades, relative to the initial development cost of a standard EIS as determined in Table 2.

With the drastic reductions in update efforts, organisations can also more readily afford to take advantage of every update that is produced by the meta-data EIS application vendor. Currently, due to the often high costs associated with updating EIS applications, many organisations may choose to skip many updates and even entire major version changes – this also has a productivity cost to the organisation which is delaying or avoiding any productivity benefits that the updated EIS application would have provided.

Meta-Data EIS to Meta-Data EIS Upgrade Savings	Number EIS Generation Changes	Relative Sum Standard EIS Cost (Initial + Upgrades) (A)	Relative Sum Meta-Data EIS Cost (Initial + Upgrades) (B)	Relative Ongoing Generations Meta-Data EIS Ratio (B/A)	Relative Meta-Data EIS Cost (Maintain Initial) (C)	Relative Same Generations Meta-Data EIS Ratio (C/A)
25%	1	450%	90%	20%	90%	20%
	2	900%	170%	19%	147%	16%
	3	1350%	251%	19%	204%	15%
	4	1800%	331%	18%	261%	15%
	5	2250%	411%	18%	319%	14%

Table 4 - Estimated Relative Organisational Multi-Generational Costs

Table 4 illustrates the relative estimated costs for an organisation's multi-generational EIS lifecycle, where an organisation may regularly change to an alternate vendor's EIS application. These estimates include the initial and ongoing upgrade costs for the standard EIS vs the meta-data EIS application, assuming a full transfer to the new EIS at each generational stage for both options. Note that an additional efficiency (25% as listed) is expected when upgrading from one meta-data EIS application to a completely different meta-data EIS application due to the ready availability of knowledge of the meta-data structures, in addition to the ability to directly access the existing meta-data EIS structure when implementing the new meta-data EIS application – effectively continuing to lower the ongoing total relative cost for the meta-data EIS application option. A final option (right side breakout) that naturally offers even lower ongoing costs is where the original meta-data EIS application is able to be maintained through the EIS generations replacing each EIS change with a major upgrade – this is not unrealistic as technology platform updates which are the typical driver for generational change are meta-data model independent and are provided by updates to the independent runtime execution engines.

These costing illustrations do not heavily factor in the potential large additional productivity savings to the user base that can be achieved by having local users readily modify the behaviour of their local meta-data EIS application for rapid local improvements, as Logic Variants. It is the author's opinion that such benefits could progressively offer an even greater magnitude of further cost savings to user organisations.

The examples provided to demonstrate the potential cost savings are a mix of comparative vendor's developments costs combined with users' indicative implementation costs. Whilst this may not be an accurate single scenario it has been useful to identify the combined relative indicative costs. In an ongoing competitive meta-data EIS application environment where there is consumer choice, any drastic development savings by vendors would generally necessarily be passed on to users after a technology bedding in period. Thus the combined vendor / user industrial base would tend to progressively approach the forecast level of savings as the technology and methodologies of the meta-data EIS applications became widespread and mature within the vendor and user organisations.

4.10 Conclusion

Chapter 2 defines the current state of the art research of Model Driven Architectures and the use of meta-data modelling as alternatives to the traditional code based development of Enterprise Information Systems. Substantial research is being undertaken in these areas, particularly in the field of Model Driven Architecture.

With the introduction of more cross platform languages, frameworks and platform compatibility, modular software development, web services, improved Computer Aided Software Engineering tools and team collaboration tools, a continual enhancement in reusability, productivity, quality and cost improvements have been observed. However, few research areas have provided contributions that effectively address fundamental and widespread efficiency optimisations for the software development lifecycle, nor the empowerment of the business users to more directly address and modify their own business practices and their use of the EIS applications.

As a solution to providing a paradigm shift in reducing EIS development and deployment costs and timescales, and in providing unparalleled flexibility to EIS

business users, I have proposed a temporal meta-data framework. The framework is used to first record the design of the EIS application in its meta-data structure, and then directly execute the meta-data EIS application from the meta-data with the runtime engine of the temporal meta-data framework, with no direct coding required, achieving greatly optimised and reduced development efforts. Business users are also provided with the ability to modify and specify their own application functionality within the meta-data EIS application without the need for specialist technical development. The framework is applicable to the wide range of business EIS applications and with appropriate extensions, potentially expandable to incorporate and replace a much wider range of more technical software development environments.

In this chapter, I have described the key design requirements and capabilities of the temporal meta-data framework based on the research issues discussed in chapter 3.

Chapter 5 describes the meta-data structure of the framework model in detail, chapter 6 describes the design of the runtime engine for the framework, chapter 7 reviews the design accelerants for a meta-data editor, and chapter 8 summarises the interfaces that can support global access and interoperability via the cloud. These chapters are used to illustrate and empirically evaluate the framework and methodology with Chapter 9 demonstrating a progressive incremental definition of the meta-data for the creation of a sample meta-data EIS application using this framework as a final validation of concept.

Chapter 5 - Instant Interaction EIS System Modeller

5.1 Introduction

One of the aims of this thesis is the capability to define meta-data EIS applications. To satisfy this requirement a model for meta-data EIS applications needs to be created that will capture all of the detailed application logic that may be required for the future execution of the meta-data EIS application.

Such a model needs to encompass all of the expected features of these applications, and necessarily needs to be mapped to the functionality to be provided by the associated temporal meta-data framework runtime execution engine.

As EIS applications tend to be defined using the multi-layered design approach, so too will the design of the meta-data EIS application incorporate the common design layers of;

- **Visual Structure Elements**: model to define the layout and functionality of the application user interface for operation by the application users.
- **Program Flow Elements**: logical operations that are performed on data or as workflow sequences as a consequence of the operation of the user interface elements.

- **Extended Data Dictionary**: model of the associated application data structures including higher level abstractions and aggregation structures.

In the following sections of this chapter, I present detailed summaries of how these layers of the meta-data EIS application can be successfully modelled for execution by the temporal meta-data framework runtime execution engine. The models are based on standard entity-relationship models to emphasise the true meta-data nature of the models. Full model designs are provided in the appendices.

Additional descriptions are also provided of the more global model aspects that support the advanced meta-data EIS application features such as; temporal execution, application adaptability, automated application deployment, and automated user education, as identified in the previous chapter.

The design models presented for the meta-data EIS application can be very complicated in areas, even with the use of a modern CASE tool to manage the model design.

In order to maximise the understanding of the model presentation to the thesis reviewer (and at times the author), the design model structure has been separated into discrete logical models concentrating on a particular function. Additional background model layers are used to model some of the more repetitive and common aspects to the models.

5.2 Common Model Elements

There are some elements of the meta-data EIS application model that are common to all layers of the model. These common elements are first described in this section and elaborated as required in the more detailed layer model descriptions.

5.2.1 Generic Distributed Temporal Meta-Data Inheritance

One such common modelling aspect is the Generic Distributed Temporal Meta-Data Inheritance which is a portrayal of the common relationships that apply to most of the meta-data model entities associated with the design of the meta-data EIS application model structures.

The background relationships are used to specify functionality that needs to apply to most of the meta-data entities such as:

- **Distributed Execution**: allow multiple distributed meta-data instances to interoperate.
- **Temporal Management**: maintaining a permanent temporal state of the meta-data.
- **Common Naming**: of meta-data entities and for descriptions.
- **User Education Notes**: user help and manuals.
- **Multi-Lingual**: allow multi-language support for the meta-data.
- **Logic Variants**: allowing user customisation of any application objects.
- **Audit Security**: tracking the changes made to meta-data definitions.

Figure 22 depicts a sample of the full relationships that need to apply to the majority of meta-data entities in the meta-data EIS application model in order to provide the above requirements. Whilst these relationships are defined within the general model design, they are not usually depicted in the functional design excerpts that are illustrated in most areas of this thesis.

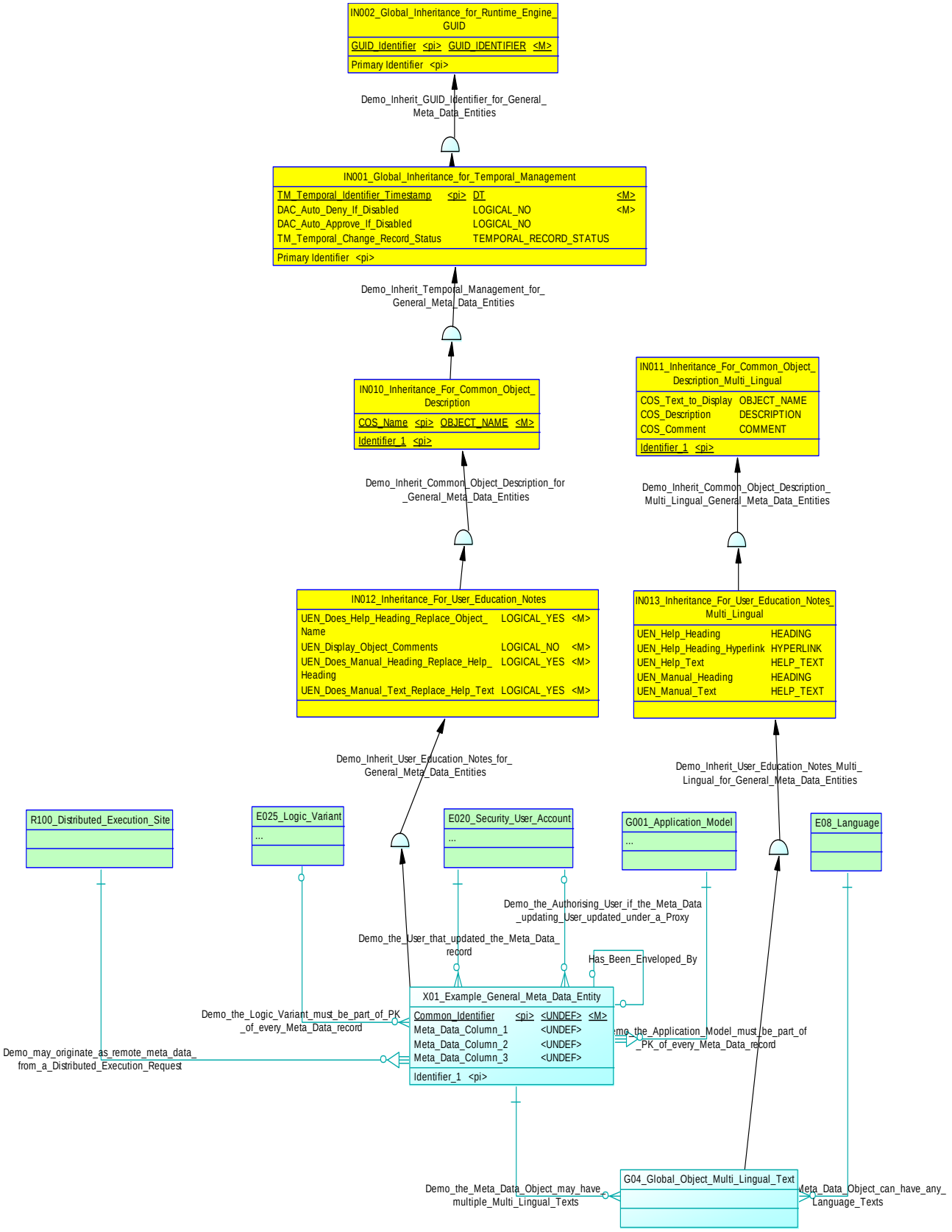


Figure 22 – Generic Distributed Temporal Meta-Data Inheritance

The Generic Distributed Temporal Meta-Data Inheritance design uses the following entities to model the required functionality:

- **Global Inheritance for Runtime Engine GUID**: is inherited to all meta-data entities. It provides a unique identifier primary key purely to manage a unique global identification for all objects.
- **Global Inheritance for Temporal Management**: is inherited to all meta-data entities. It provides the temporal alternate primary key and status record required to identify all temporal records – the status will be one of:
    - **Null**: for the current record.
    - **Deleted**: for the most current record to mark the entire record set as deleted.
    - **Superseded**: to identify non-current records.
- **Inheritance For Common Object Description**: is inherited to all meta-data entities. It provides the naming alternate primary key.
- **Inheritance For User Education Notes**: is inherited to all meta-data entities. It provides control options for how the corresponding user help and manual textual data is used.
- **Distributed Execution Site**: identifies a distributed meta-data instance to allow interoperation. This is modelled as a relationship from all meta-data entities to identify the originating instance.
- **Logic Variant**: is a designated identifier to group all of the logic changes together into a practical set. This is modelled as a relationship from all meta-data entities to identify what variants have been defined for a meta-data object.
- **Security User Account**: is the list of Users that are defined in the application runtime execution environment for the meta-data. This is modelled as two relationships from all meta-data entities to track the users that create or modify the meta-data as well as any authorised proxy user that may be operating for another user.
- **Application Model**: is the high level identifier of the application as modelled in the meta-data EIS application. This is modelled as a dependant relationship from all meta-data entities to provide the

Application Model identifier as part of the primary key to group all of the application's meta-data objects.

- **Example General Meta Data Entity**: is a sample meta-data entity to represent the relationships that most of the other the meta-data entities will have as background relationships. E.g. Canvas, View Columns etc.
- **Inheritance For Common Object Description Multi Lingual**: is inherited to the Global Object Language meta-data entity. It provides the meta-data object naming and description information for any language.
- **Inheritance For User Education Notes Multi Lingual** is inherited to the Global Object Multi Lingual Text meta-data entity. It provides the meta-data object user help and manual textual information for any language.
- **Language**: is a list of languages used for localisation options. This is modelled as a relationship from the Global Object Language Text meta-data entity to identify the specific language for a meta-data object's multi-lingual definition.
- **Global Object Multi Lingual Text**: maintains the above inherited text data for each meta-data object on a per language basis to provide a multi-lingual solution for any meta-data EIS application.

To identify all of these relationships for each instance of a meta-data entity in the model would create a very confusing model design. Accordingly these relationships are defined in separate repetitive models allowing the key meta-data entities to be presented for their major functions. The full model design is listed as an appendix to this thesis.

5.2.2 Generic Distributed Temporal Data Inheritance

The meta-data EIS application can operate with any third party database schema that is supported by the runtime engine. Typically such an externally created database would consist of only the key data entities although depending on the origins of the entity, there may be surrounding entities and/or attributes that were defined to provide specialised database management functions by a third party system.

For optimum functionality, the meta-data EIS application also specifies additional entities and attributes that would be created around any defined data entity to ensure that all of the available beneficial features of the meta-data EIS application are available, such as;

- **Distributed Execution**: allow multiple distributed meta-data instances to interoperate.
- **Temporal Management**: maintaining a permanent temporal state of the data.
- **Multi-Lingual**: allow additional multi-language support for the data that can allow alternate translations of the data.
- **Audit Security**: tracking the changes made to the data.

Figure 23 depicts a sample of the full relationships that need to apply to any data entities used by the meta-data EIS application in order to provide the above additional data management capabilities.

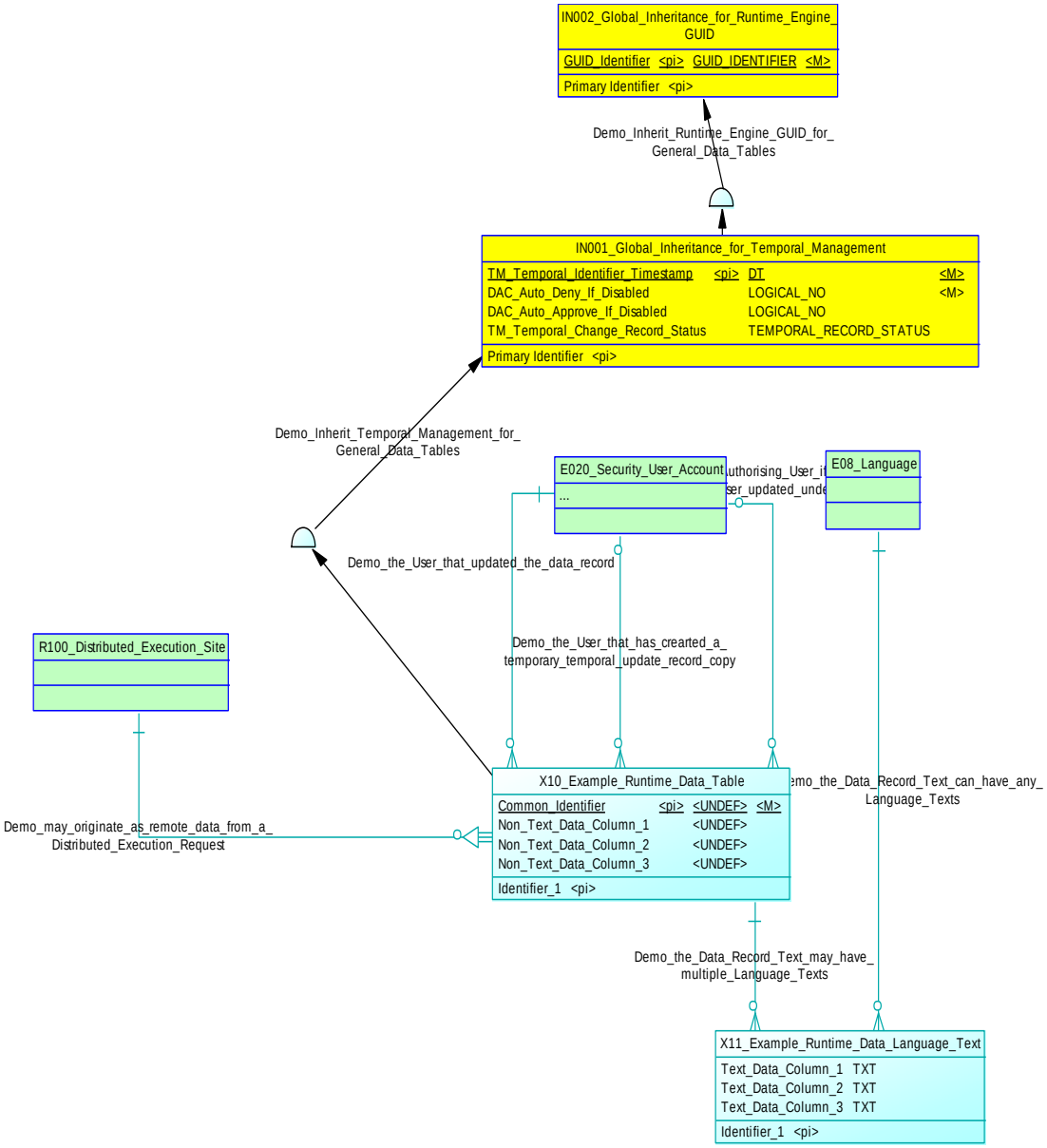


Figure 23 – Generic Distributed Temporal Data Inheritance

The Generic Distributed Temporal Data Inheritance design uses the following entities to model the required functionality:

- **Global Inheritance for Runtime Engine GUID**: is inherited to all data entities. It provides a unique identifier primary key purely to manage a unique global identification for all data.

- **Global Inheritance for Temporal Management**: is inherited to all data entities. It provides the temporal alternate primary key and status record required to identify all temporal records – the status will be one of:
    - **Null**: for the current record.
    - **Deleted**: for the most current record to mark the entire record set as deleted.
    - **Superseded**: to identify non-current records.
- **Distributed Execution Site**: identifies a distributed meta-data instance to allow interoperation. This is modelled as a relationship from all meta-data entities to identify the originating instance.
- **Security User Account**: is the list of Users that are defined in the application runtime execution environment. This is modelled as two relationships from all data entities to track the users that create or modify the data as well as any authorised proxy user that may be operating for another user.
- **Language**: is a list of languages used for localisation options. This is modelled as a relationship from the above data entity to identify the specific language for a data record's multi-lingual translation.
- **Example Runtime Data Table**: is a sample data entity to represent the relationships that all of the other the data entities will have as background relationships. Note that all text attributes should not be defined in this entity – all text based attributes must be defined in the following entity in order to provide multi-lingual data support
- **Example Runtime Data Language Text**: is a sample data entity to represent the distinction between the non-text attributes (defined in the above entity) and the text attributes that are defined in this entity and thus available for alternate language translations. This is a modelled as a relationship from the above data entity to maintain the text data for each data record on a per language basis to provide a multi-lingual translation solution.

It may not be possible to apply these entities, attributes and relationships to all external data so the additional features of the meta-data EIS application may not be able to be provided for such data that is managed externally. However, all data that is

defined with these entities, attributes and relationships will receive the full temporal, multi-lingual and audit benefits provided by the meta-data EIS application.

5.2.3 Application Model

The Application Model entity represents the ultimate common placeholder identifier for application models that are installed or available i.e. defined within the current meta-data repository for execution by the runtime engine.

An Application Model may have hundreds or even thousands of dependent elements that collectively define the layout, logic and data of the application. It is synonymous with an EIS application as a whole.

The Application Model object acts as a pointer to the starting logic of the application although initially, the runtime engine will need to resolve who the current user is, either by a specific login request, or alternatively as identified via start up parameters to automatically login as a specified user or to identify the currently operating system user.

Following user validation, the Application Model will either execute its associated initial logic sequence if it is defined or proceed to execute its first Canvas if it is defined – one of these must be defined for any further logic to proceed. The remainder of the meta-data EIS application logic will flow from either or both of these elements as defined.

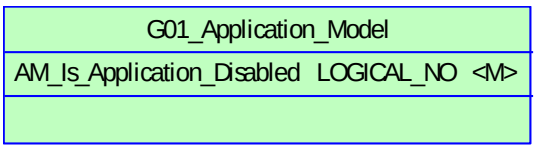


Figure 24 – Application Model Entity – Base Model Component

Figure 22 illustrates the basic conceptual structure of the Application Model entity. The conceptual model schema is very simple and represents mainly a dependent placeholder to be inherited to all other model elements.

All other elements of the meta-data EIS application model will be related to the Application Model entity as a parent relationship to identify all elements of the meta-data EIS application and allow multiple meta-data EIS application model to co-exist.

5.3 Visual Structure Elements

Every GUI application will consist of collections of common UI Objects such as text, buttons, data grids etc grouped together in logical arrangements on the screen in collectives commonly referred to as forms or pages. The availability of the UI Objects is dependent on the functionality provided by the supporting framework

The meta-data EIS application model uses similar user interface design metaphors and objects to other common EIS applications and integrated development environments – the primary difference being that in the meta-data EIS application the existence and relationships of the visual objects is maintained in a readily modifiable model structure rather than as a compiled object, although there are additional advantages provided such as advanced linking and association features of the meta-data EIS application which can provide automatic object linking, function access and direct workflow functionality.

Figure 25 illustrates a fundamental relationship between the visual elements of the meta-data EIS application (denoted in the diagram as Structural Elements) and any defined logic processing (denoted in the diagram as Events). The visual elements can invoke the execution of any logic processing, and the logic processing can also invoke the activation of any visual elements. In the meta-data EIS application logic processing is defined and executed as nested functions with the activation of visual elements processed as a simple function pre-defined for each defined visual element. Further detail on the features of the logic processing are discussed in 5.4 Program Flow Elements.

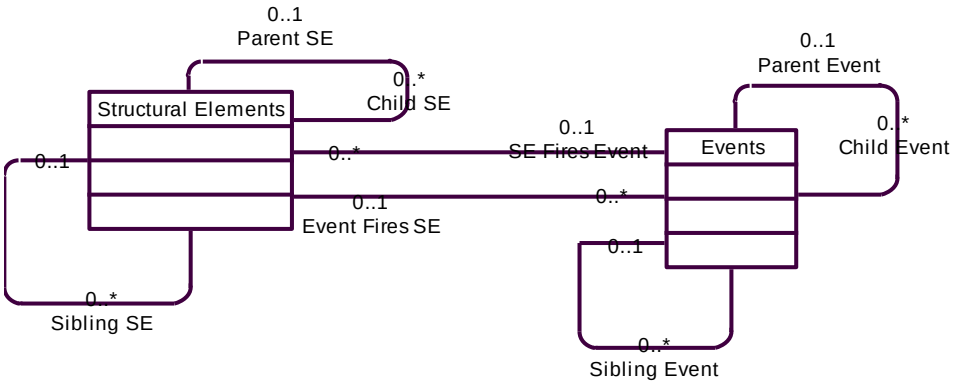


Figure 25 – Relationship Between the Visual Element Structure and Logic Processing Events

The visual structure of the meta-data EIS application model is first described via the basic structure of the user interface elements and their relationships, followed by a description of the higher level user interface objects that are defined to provide compound functionality.

The expanded logical structure of all visual structure elements is listed fully in the appendices.

5.3.1 Fundamental User Interface Model

The basic organisational layout of the user interface structure in the meta-data EIS application model is composed of the following elements:

- **UI Inheritance Objects**: common object sizing and location attributes.
- **Canvas**: has similarities to what is often referred to as a form and is composed of panels.
- **Visual Application Structure**: is used to optionally organise a hierarchical list of Canvas entities to simulate any desired implied management structure such as modules etc.
- **Navigation Panel**: can be defined similarly to common menus or toolbars to provide fast selection of and navigation to a canvas.
- **Navigation Panel used on Canvas**: is used to share Navigation Panel objects between Canvas objects.

- **Navigation Panel Item**: Are the individual shortcut items to a Canvas that compose a Navigation Panel to provide the Menu and/or Toolbar functionality.
- **Freeform Panel**: are the basic collectives of user interface elements referred to as UI Objects, grouped together into logical collections for display and operation.
- **Freeform Panel used on Canvas**: is used to share Freeform Panel objects between Canvas objects.
- **UI Object**: are the references to the each of the individual UI Objects such as; text, buttons, data grids etc that are available.
- **UI Object used on Freeform Panel**: is used to share UI Objects between Freeform Panel objects.
- **UI Alignment Rule**: define the layout relationships between UI Objects and their host Freeform Panels, or between Freeform or Navigation Panels and their host Canvas entities. See 5.3.3.2.2 UI Alignment Rule .
- **UI Alignment Collation**: helps define the final result of the layout relationships options between UI Objects and their host Freeform Panels, or between Freeform or Navigation Panels and their host Canvas entities. See 5.3.3.2.3 UI Alignment Collation .

Figure 26 illustrates the basic entities and relationships of the visual structure of the meta-data EIS application model.

Each entity is discussed in more detail in the following sections including a brief discussion of each of the defined UI Objects.

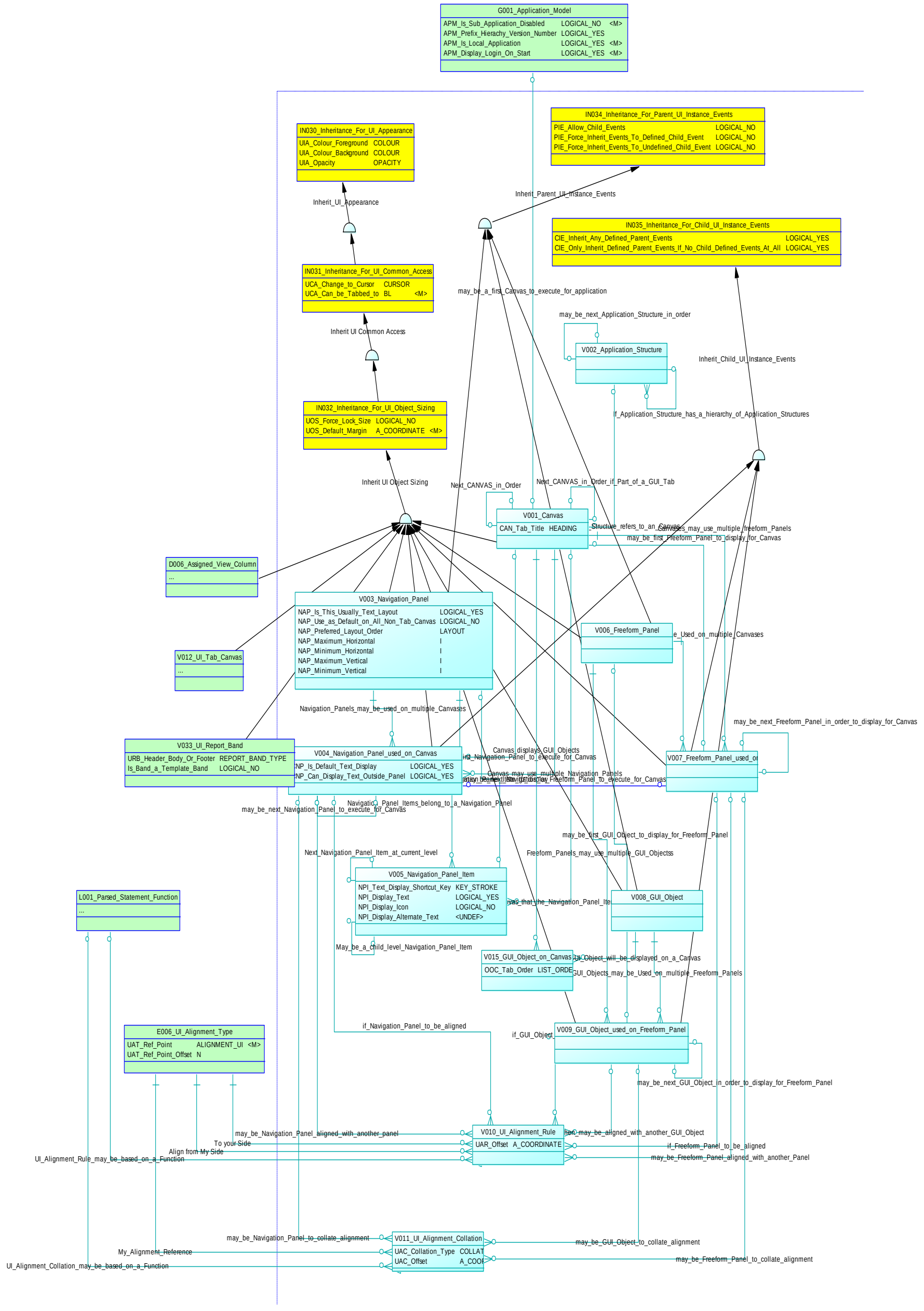


Figure 26 – Primary Visual Object Structure

5.3.1.1 UI Inheritance Objects

All visual entities will inherit the common characteristics of the visual structure elements. As listed in Figure 26 these inherited characteristics include:

- **Inheritance For UI Appearance**: basic object display appearance such as colours and opacity.
- **Inheritance For UI Common Access**: specify the tabbing control options for an object.
- **Inheritance For UI Object Sizing**: specifies the basic sizing and location of the object and its resizing options.

5.3.1.2 Canvas

The Canvas is a high level grouping entity, with many similarities to what is often referred to as a form or page in other development environments. Each Application Model may be composed of many Canvas objects of which one Canvas may be designated as the initial Canvas to be displayed and executed.

Multiple Canvas objects may also be defined as a UI Tab object, with each tabbed page of the UI Tab as a separate Canvas.

A Canvas is composed of panels which may be any number of Navigation Panel and Freeform Panel objects. Each panel can be defined with positional inter-relationships to other neighbouring panels that can aid in preserving a good user interface appearance during any dynamic resizing of users' screens.

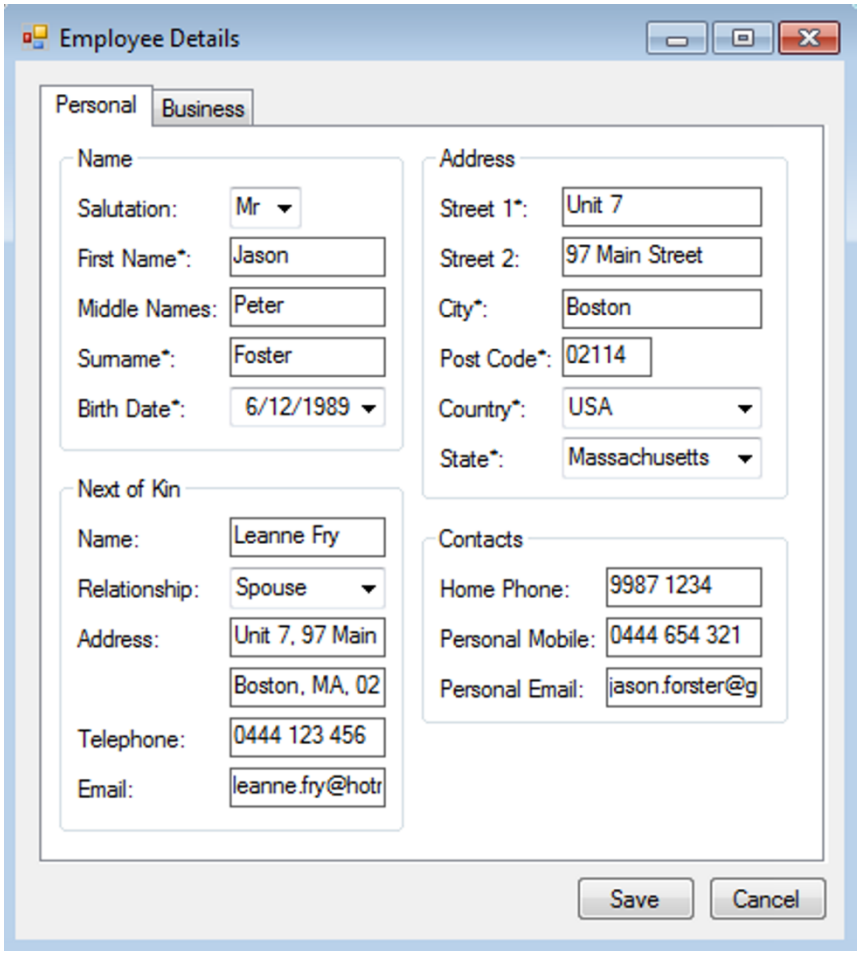


Figure 27 – Example Canvas with Multiple Panels

Figure 27 illustrates an example of a **Canvas** in a meta-data EIS application with multiple component Panels for each of the separate data entry areas, as well as a UI Tab which controls multiple Canvases.

5.3.1.3 Application Structure

The Application Structure is not actually a visual element, rather it is an application management helper object that can be optionally used to organise a hierarchical list of Canvas entities.

Such a list can be used to simulate any desired or implied management structure of the meta-data EIS application such as sub-applications or modules etc. A primary use of the Application Structure would be to organise the Canvas objects into logical groupings or modules that clearly identify major functional areas of the application. E.g. in an Accounting application the highest level structure may be General Ledger,

Accounts, Payroll, Purchasing, Invoicing, with further sub-levels defining the individual Canvas objects defined within each functional area.

Initially, the Application Structure provides a grouping aid to progressively organise the structure of the meta-data EIS application as the components are iteratively defined. As the definition of the meta-data EIS application matures, the Application Structure can then also provide a source of data for Navigation Panel objects to act as menus and provide fast access to the Canvas objects.

5.3.1.4 Navigation Panel

The Navigation Panel can be defined similarly to common menus or toolbars to provide fast selection of and navigation to Canvas objects.

The fundamental distinction is that menus are text based and similar to the drop-down menus experienced in typical GUI applications, while toolbars are icon based. Depending on the runtime environment (e.g. thick vs thin functionality) menus would offer drop-down functionality, while toolbars may be capable of docking and floating,

Navigation Panels may be accessed by and shared between any Canvas objects as defined in the Navigation Panel used on Canvas entity.

5.3.1.5 Navigation Panel used on Canvas

The Navigation Panel used on Canvas is not a visual element, it is an application association object that allocates defined Navigation Panel objects to Canvas objects, thus allowing the reuse and sharing of Navigation Panel objects between Canvas objects.

This allows for options such as e.g. maintaining a common menu for many Canvas objects, defining different toolbars for different Canvas objects, and also removing all navigation if a maximum of screen display space was required on a Canvas.

5.3.1.6 Navigation Panel Item

The Navigation Panel Item objects are the individual shortcut items to a Canvas that compose a Navigation Panel to provide the Menu and/or Toolbar functionality.

Each shortcut is defined with its textual and/or icon representation as the display may be different for different application contexts. The ultimate target of a Navigation Panel Item is any defined Canvas object.

5.3.1.7 Freeform Panel

The Freeform Panel represents the key groupings of UI Objects into logical and useable display components that will be presented as part of the operational user interface to the users.

Each UI Object can be defined with positional inter-relationships to other neighbouring UI Objects that can aid in preserving a good user interface appearance during any dynamic resizing of users' screens, similar to the Panel positional inter-relationships within the Canvas.

Freeform Panels may be accessed by and shared between any Canvas objects as defined in the Freeform Panel used on Canvas entity.

Figure 27 illustrates an example of multiple Panels on a Canvas.

5.3.1.8 Freeform Panel used on Canvas

The Freeform Panel used on Canvas is not a visual element, it is an application association object that allocates defined Freeform Panel objects to Canvas objects, thus allowing the reuse and sharing of Freeform Panel objects.

This allows any logical grouping of visual elements as represented by a Freeform Panel to be replicated and reused at key areas of an application without the effort and inherent risks that duplication of the visual elements may introduce.

A key example of such replication may be where Freeform Panels are replicated from normal user entry screens onto specialist wizard-style workflow screens that can more readily direct the preferred logical entry and processing flow.

5.3.1.9 UI Object

UI Objects are the individual and very common visual artefacts that are familiar to users of modern GUI systems. E.g. buttons, drop-down lists, tree controls etc. The meta-data EIS application does not prescribe a great deal of variation in the use of and definition of these objects, other than their internal modelling structure, and how the UI Objects can readily take advantage of advanced linking and association features of the meta-data EIS application such as the automatic object linking, function access and direct workflow functionality.

UI Objects are classified into two groups:

- **Basic User Interface Objects**: are the most common and simplest user interface objects such as; buttons, drop-down lists, text box. These are further discussed in 5.3.2.2 Basic User Interface Objects :
    - **UI Line**: simple line drawing.
    - **UI Video**: display video image.
    - **UI Image**: display static image.
    - **UI Button**: button control to click for an action.
    - **UI Rectangle**: simply draws a square or rectangle.
    - **UI Ellipse**: drawing a circle or ellipse.
    - **UI Text**: display text.
    - **UI Text Box**: prompt for the entry of user text input.
    - **UI Slider**: controls the value of a variable.
    - **UI Selection:** manages lists and merges the functionality of Check Box, Combo Box, Drop Down Box, List Box and Radio Buttons into the chosen format.
- **Advanced User Interface Objects**: provide more advanced functionality such as data access or are compound objects that combine multiple functionality such as; data grids, tab controls. These are further discussed in 5.3.2.3 Advanced User Interface Objects:
    - **UI Tree**: displays data in an expandable / collapsible tree representation.
    - **UI Data Grid**: links data from view Tables to a high functionality grid representation for visual review.
    - **UI Cross Tab**: provides a tabular aggregation of data as an analysis tool.
    - **UI Report**: provides similar functionality to panels and data grids, formatted as bands to support common report functionality.
    - **UI Tab**: allows multiple panels to be managed within a single element like a tabbed binder.
    - **UI Chart**: provides common graphical collation and charting.

5.3.1.10 UI Object used on Freeform Panel

The UI Object used on Freeform Panel is not a visual element, it is an application association object that allocates defined UI Objects to Freeform Panel objects, thus allowing the reuse and sharing of UI Objects between multiple Freeform Panels.

This allows any individual visual elements as represented by a UI Object to be replicated and reused at key areas of an application without the effort and inherent risks that duplication of the visual elements may introduce.

5.3.2 Modeled User Interface Objects

The meta-data EIS application model uses similar user interface design metaphors and objects to other common EIS applications and integrated development environments, which are denoted as UI Objects – the primary difference being that in the meta-data EIS application the existence and relationships of the visual objects is maintained in a readily modifiable model structure rather than as a compiled object, although there are additional advantages provided such as advanced linking and association features of the meta-data EIS application which can provide automatic object linking, function access and direct workflow functionality.

UI Objects are classified into two groups:

- **Basic User Interface Objects**: are the most common and simplest user interface objects such as; buttons, drop-down lists, text box.
- **Advanced User Interface Objects**: provide more advanced functionality such as data access or are compound objects that combine multiple functionality such as; data grids, tab controls.

Figure 28– Basic UI Object Model illustrates the basic entities and relationships of the UI Object visual entities of the meta-data EIS application model.

Each UI Object entity is discussed in more detail in the following sections.

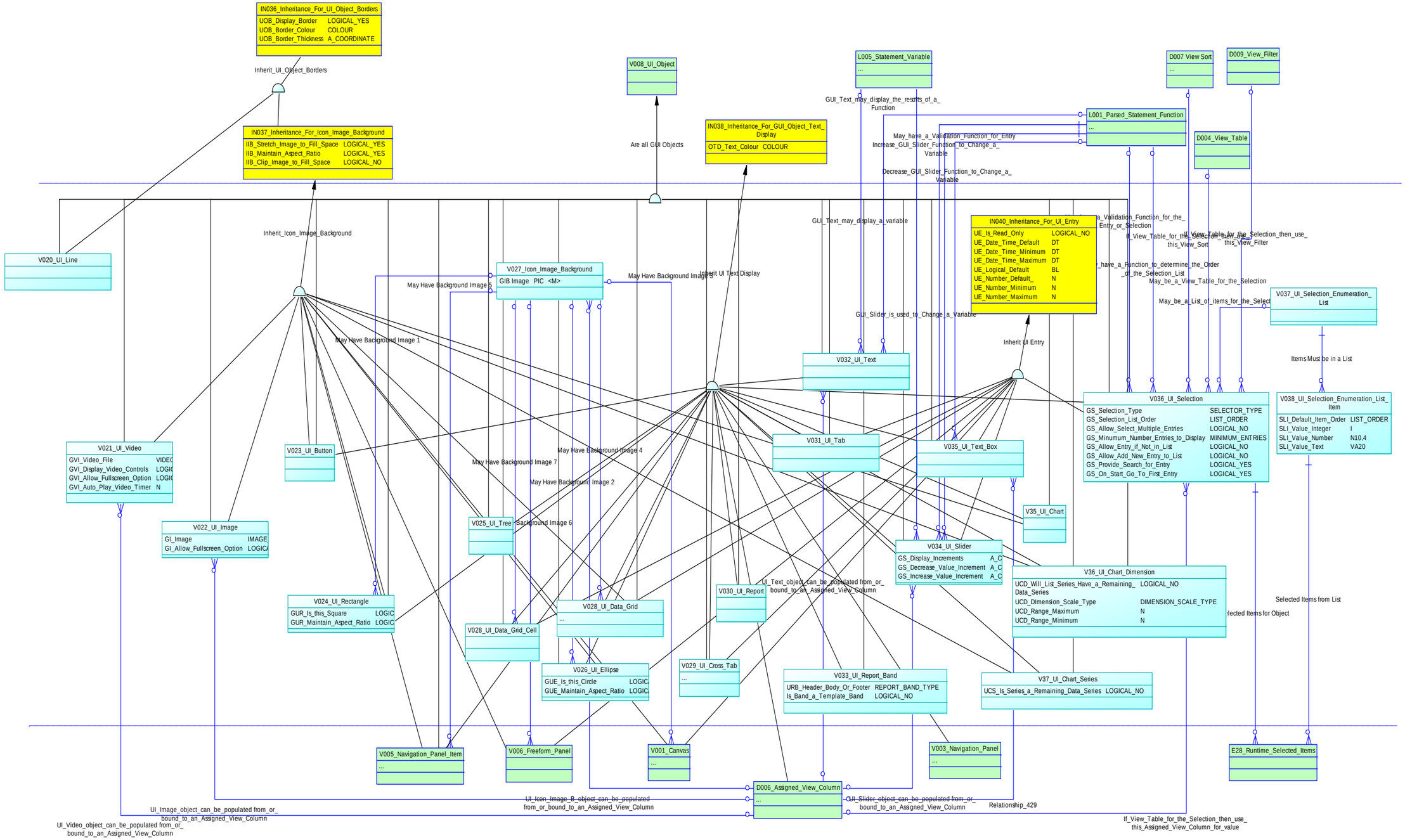


Figure 28 – Basic UI Object Model

5.3.2.1 UI Object Inheritance Objects

All UI Objects will inherit some common characteristics. As listed in Figure 28– Basic UI Object Model these inherited characteristics include:

- **Inheritance For UI Object Borders**: object display appearance for any surrounding border.
- **Inheritance For Icon Image Background**: specify how a background image may be displayed if available for an object.
- **Inheritance For UI Object Text Display**: object display appearance for any displayed text of an object.
- **Inheritance For UI Entry**: specifies any entry masks and basic validation for objects requiring user entry.

5.3.2.2 Basic User Interface Objects

These are the most common and simplest user interface objects such as; buttons, drop-down lists, text box. They are the fundamental building block objects of the user interface interactions and generally simple atomic function objects with limited functionality and operation.

5.3.2.2.1 UI Button, Text and Text Box

These objects are very similar to standard GUI operation:

- **UI Button**: is a standard button control which can be allocated functions to execute based on various actions, most commonly such as clicking.
- **UI Text**: object simply displays static text, the value of a Variable, or the results of a Function.
- **UI Text Box**: object prompt for the entry of user text input and may reference a Function for entry validation.

5.3.2.2.2 UI Selection and Enumeration

The UI Selection object manages lists and merges the functionality of the Check Box, Combo Box, Drop Down Box, List Box and Radio Buttons into the chosen format.

While the particular object type e.g. Combo Box can be selected specifically, the actual object requirements are alternatively modelled for dynamic allocation of a UI object type and format.

Additionally, the UI Selection integrates into the Selection List Enumeration object management which manages all available lists and selections, further offering optional Enumerated values based on the selection.

5.3.2.2.3 UI Slider

The UI Slider is a visual slider object that fundamentally controls the value of an assigned variable by its operation. The Decrement and Increment actions of the UI Slider can be specified as values or based on the results of Functions.

5.3.2.2.4 UI Image, Video and Audio

The UI Image is used to display static images whilst the UI Video displays video images. Audio is not specified as a specific object although can be assigned as an object hover function.

5.3.2.2.5 UI Line, Rectangle and Ellipse

These objects are very similar to standard GUI operation:

- **UI Line**: simply draws a line between coordinates.
- **UI Rectangle**: draws a coordinate based rectangle which can be fixed to a square if specified, and may display an image or text.
- **UI Ellipse**: draws a coordinate based ellipse which can be fixed to a circle if specified, and may display an image or text.

5.3.2.3 Advanced User Interface Objects

These user interface objects provide more advanced functionality such as data access or are compound objects that combine multiple functionality such as; data grids, tab controls.

They represent significant internal complexity and provide advanced functionality to minimise the need for additional user logic definitions.

Figure 29 illustrates the basic entities and relationships of the advanced visual entities of the meta-data EIS application model.

Each entity is discussed in more detail in the following sections.

{This page intentionally blank}

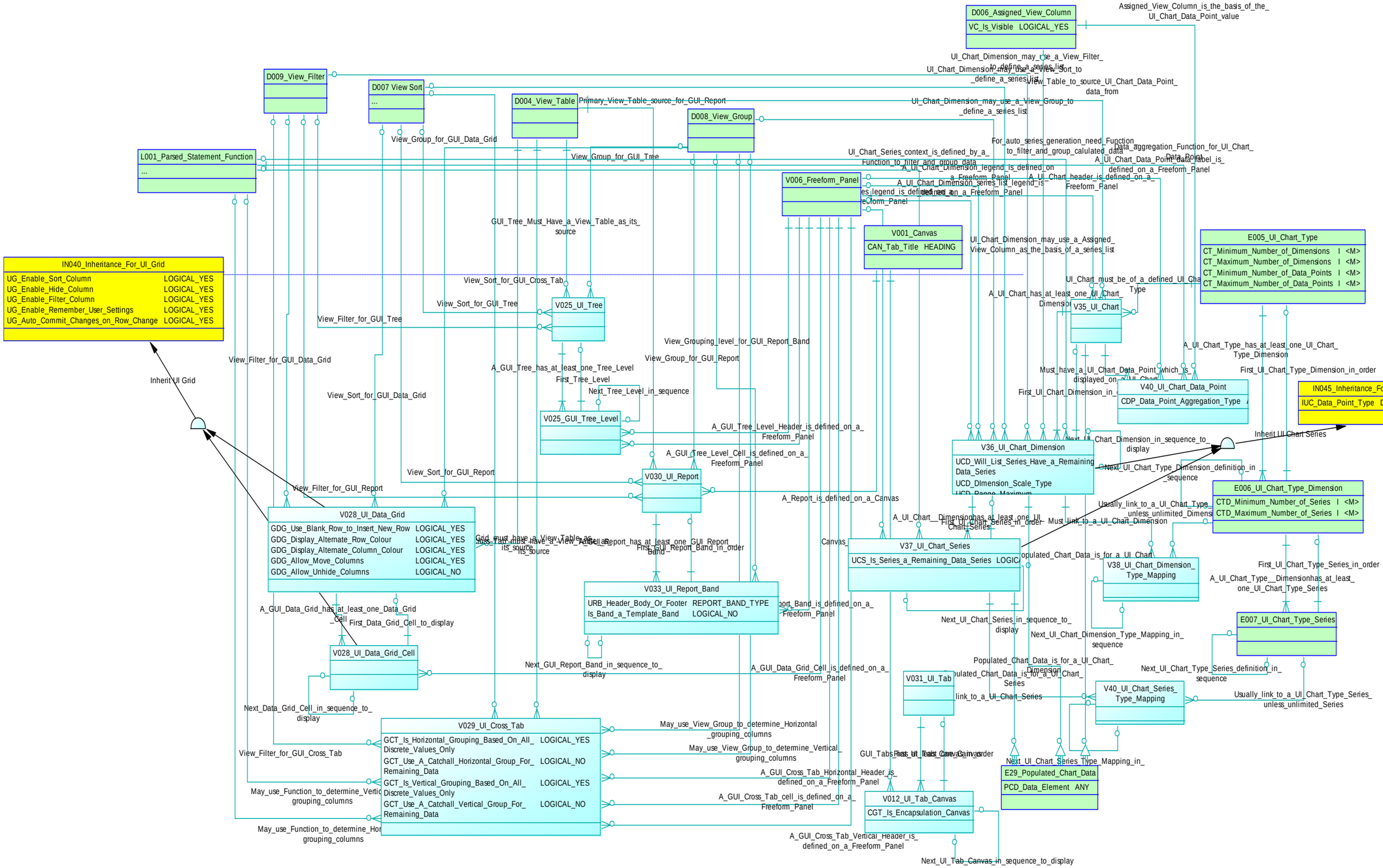


Figure 29 – Advanced UI Object Model

5.3.2.3.1 UI Data Grid

Data grids are commonplace in modern GUI systems although implemented functionality varies widely.

The key aspect of the UI Data Grid is its direct linkage to key data dictionary elements of the meta-data EIS application model:

- **View Table**: an abstracted data set definition of View Columns which can be further abstracted for personalised or localised access, which also defines the column display order.
- **View Sort**: specified multi-level sort criteria to apply to the View Table.
- **View Filter**: specified data filter criteria to apply to the View Table.

Individual View Columns from the specified View Table of the UI Data Grid can also be assigned dynamic links to other objects such as Variables and UI Objects that will offer optional 2-way synchronisation between these objects and the relevant UI Data Grid / View Table components i.e. the underlying columns. This allows for simple interaction between the meta-data EIS application objects and the internal data, and readily and quickly facilitates such features as Master-Detail or Parent-Child style forms.

5.3.2.3.2 UI Tree

Tree controls are also relatively commonplace in modern GUI systems to display data in an expandable / collapsible tree representation.

Again the key aspect of the UI Tree is its direct linkage to key data dictionary elements of the meta-data EIS application model; View Table, View Sort and the View Filter.

Additionally, the UI Tree is further specified by the use of the View Group which defines additional data grouping criteria to apply to the View Table and hence form the defined branch definition for the UI Tree.

5.3.2.3.3 UI Tab

Tab controls are another common GUI control allowing multiple form styles to be associated together in a minimised screen real estate configuration.

The UI Tab is modelled as a set of Canvas objects in series where each Canvas (and its component Freeform Panels and/or Navigation Panels) will be displayed as a separate tab of the UI Tab.

An optional Encapsulation Canvas can be specified to provide the overall default dimensions for the UI Tab.

5.3.2.3.4 UI Report

Reporting functionality is often provided courtesy of an embedded or otherwise third party specialist reporting application which can provide generic and best of breed reporting capability to an EIS.

The meta-data EIS application exposes its data via; database views, direct SQL host database access, and via its global XML access specification, thus readily supporting such access by external third party tools.

However, the fundamental general visual modelling for a report is not dis-similar from the modelling for a form or Canvas, and in the context of the meta-data EIS application the key difference is that a UI Report can be divided into multiple UI Report Bands corresponding to the required grouping levels of the report.

The UI Report is structured as:

- The primary data source for the report is defined by; View Table, View Sort and the View Filter.
- A Canvas is used to capture the UI Report visual design.
- Each UI Report Band specifies a level of the defined View Group as its grouping criteria. A Freeform Panel is used to capture the UI Report Band visual design. UI Objects linked to the required atomic View Column will represent the requirement to display any repeating record data within that band. UI Objects linked to the required atomic View Column and defined with an appropriate aggregation function will be evaluated according to the aggregation function for data within that band.
- The concept of embedded or sub-reports is also facilitated as a UI Report is a sub-type of UI Object and hence can be incorporated into the UI Report design in any band. Access to data from within the sub UI Report can be via direct access to any of the component objects of that sub UI Report from an object within the UI Report.

5.3.2.3.5 UI Chart

Charting functionality is also typically provided courtesy of an embedded or otherwise third party specialist reporting application which can provide generic and best of breed reporting capability to an EIS, however it is a relatively simple extension to the core model structure.

The UI Chart is structured as:

- UI Chart Type: captures the definition of any pre-defined chart types.
- UI Chart Series: refer to the defined data sources for each chart series.
- UI Chart Data Point: can define data aggregation functions on the source data.
- UI Chart Dimensions: relates chart series to pre-defined chart dimensions for a chart type that determine the presentation and layout.
- Type Mappings: can apply to chart series and dimensions to modify the display of the data.

5.3.2.3.6 UI Cross Tab

Cross Tab functionality is used to provide a tabular aggregation of data as an analysis tool by pivoting the columnar data into pseudo columns based on a defined aggregation and calculation criteria.

The UI Cross Tab is structured as:

- The primary data source for the report is defined by; View Table, View Sort and the View Filter.
- The primary grouping columns (which are displayed as the leftmost columns) are defined by a primary View Group.
- The secondary sub grouping columns (which are displayed as the rightmost columns) are defined by either:
    - Specifying that the discrete values as a result of a single Function should define the sub groupings, or
    - Defining a secondary View Group to define these columns (which may be individually Function based) and may require specifying whether an additional automatic sub grouping column should be used to capture any remaining data that does not satisfy any sub grouping Function result.

- The contents of each UI Cross Tab cell (as the intersection of each primary grouping row with the sub grouping columns) is defined as a Freeform Panel and displays its component UI Objects as defined.

5.3.3 User Interface Management

Apart from the basic definition of each of the user interface elements there are several user interface management concepts that have also been modelled which will provide additional advanced and configurable features for the meta-data EIS application:

- **Automatic UI Generation**: defines the basic automatic program flow due to the visual structure elements.
- **Dynamic UI Alignment**: allows UI display objects to link their positioning to other similar UI display objects for dynamic positioning and resizing.
- **Visual Structure Element Events**: are defined and available for each of the defined visual element types and can be defined to invoke other visual elements and/or call functions to execute.
- **Visual Element Function Call**: allows the definition of manual program flow to visual structure elements by allowing any function to invoke a visual structure element as part of its definition.

Each concept is discussed in more detail in the following sections.

5.3.3.1 Automatic UI Generation

A key aspect of the meta-data EIS application is that the defined visual structure elements will be automatically instantiated by the runtime engine upon model execution.

There is the additional capability to define additional logic in functions including the ability to define and execute a completely non-visual meta-data EIS application. We can also define variants to any of the existing visual and non-visual application logic that can be assigned as alternate logic routes.

However, notwithstanding the above potential changes, the following generation rules will normally be executed based on only the basic definitions of the visual structure elements, and subject to the user having appropriate authorisation to access the objects in the runtime security definitions:

- **Application Model**: following any user login and verification, all Canvases (commencing with and refocussing to the Application Model's defined first Canvas) will be instantiated.
- **Canvas**: for each instantiated Canvas, all Panels (whether Freeform or Navigation Panels) as listed for that Canvas in Navigation Panel Used On Canvas and/or Freeform Panel Used On Canvas will be instantiated in order.
- **Navigation Panels**: will be composed of the listed Navigation Panel Items and displayed according to the textual or icon layouts.
- **Freeform Panels**: will be composed of any UI Objects that are listed for that Freeform Panel in GUI Object Used On Freeform Panel and will be instantiated in order.
- **UI Object**: each listed UI Object will be instantiated in order, including internal access to any associated data sources.
- The final sizing of visual structure elements (Canvas, Navigation Panel, Freeform Panel, UI Object) will initially be based on the available resolution of the application window as determined by the runtime engine, the default sizings of each UI Object and associated UI Alignment Rules will determine Freeform Panel sizings, the Navigation Panel Item rendering and options will determine the Navigation Panel sizings, the final Panel sizings and associated UI Alignment Rules will determine final Canvas sizing.
- Dynamic resizing of any visual objects as a result of changed screen resolution, or any permitted manual resizing by users will be determined by the sizing options of the objects and any associated UI Alignment Rules.
- Access to data sources will occur automatically via the associated data source based UI Objects (UI Data Grid, UI Report, UI Chart and UI Tree) and any subsequent data updates according to allowed changes via any linked UI Objects to the data source based UI Objects that permit the data changes.
- Throughout the application execution Canvases are opened to provide additional application functionality and closed as required.

- Choosing an exit application option will close all open Canvases and exit the application.

5.3.3.2 Dynamic UI Alignment

This feature allows certain UI display objects to link their positioning to other similar UI display objects to initially aid with final UI object positioning, and to then allow for dynamic re-positioning and re-sizing of the objects during runtime.

The allowable objects that can utilise the UI alignment options are:

- **Panels**: any Freeform or Navigation Panel can be linked to any other Freeform or Navigation Panel in the same Canvas.
- **UI Object**: any UI Object can be linked to any other UI Object within the same Freeform Panel.

The dynamic UI alignment options include the following features between the similar object types:

- **UI Alignment Type**: are the defined outer display limits of each object type to act as the reference points for alignment.
- **UI Alignment Rule**: are the defined alignment rules between the similar objects. There may be multiple rules for the same alignment type.
- **UI Alignment Collation**: as there may be multiple UI Alignment Rules for a particular UI Alignment Type of an object, then there needs to be a determination of the collated outcome of these rules for the final position.

Each entity is discussed in more detail in the following sections.

5.3.3.2.1 UI Alignment Type

The UI Alignment Type defines the available visual object reference points from which to choose to anchor other visual objects to or from.

At the simplest representation the options include the basic orthogonal edges of visual objects such as: Left, Right, Top, Bottom, and object centres. The options can be readily expanded to include reference point options for more advanced visual objects and layouts such as polygonal or radially based relationships, by adding the new definitions.

5.3.3.2.2 UI Alignment Rule

The UI Alignment Rule is not a visual element, and is a visual layout helper object used to define the alignment rules between visual elements in terms of the relationship between each of the defined objects' and their selected UI Alignment Type. The rule is completed by specifying either an offset or a function to calculate the required offset between the reference edges.

There may also be multiple rules for and between objects, including multiple rules regarding a particular UI Alignment Type for an object. E.g. the Left of object A should be X points to the Right side of object B; the Left of object A should be Y points to the Right side of object C. The final resolution of these multiple rules is performed with a UI Alignment Collation rule.

Where an object needs to link to the host container itself (the host of a UI Object is its Freeform Panel, and the host of a Panel is its Canvas) then the latter alignment relationship is left blank. E.g. the Top of object A should be Z points from the Top side of a null object, would align the object from the top of its host container.

The UI Alignment Rule can be used to provide greater flexibility and automation of the dynamic layouts of visual objects, particularly when:

- Finalising the initial layout of the user interface screen and avoid micro manual alignment of objects.
- The user is displaying the user interface screens using resolutions different from the originally defined default visual layout.
- The user is resizing user interface screens during a session.
- The user may be electing to modify their own visual representations of the screens including removing or adding visual objects and adjusting their style and sizing to personal or local requirements (where authorised).

5.3.3.2.3 UI Alignment Collation

The UI Alignment Collation is not a visual element, and is another visual layout helper object used to finalise the layout for each object for each defined UI Alignment Type, where there may be multiple UI Alignment Rules defined for a visual object and the same UI Alignment Type.

Typically, the Maximum might be chosen for any individual UI Alignment Rule to avoid the overlay of objects, or alternatively the Minimum or specify a function to

calculate the final UI Alignment Collation. An additional Offset can also be optionally applied to that dimension to complete the calculation.

5.3.3.3 Visual Structure Element Events

Events are defined and available for each of the defined visual element types and can be defined to invoke other visual elements and/or call functions to execute.

Additionally, local functions can be defined which can be executed to determine if these invocations or executions should occur, thus providing an additional level of control for individual instances of the objects and of any commonly accessed functions.

Figure 30 provides an extract of the conceptual design for the event processing design.

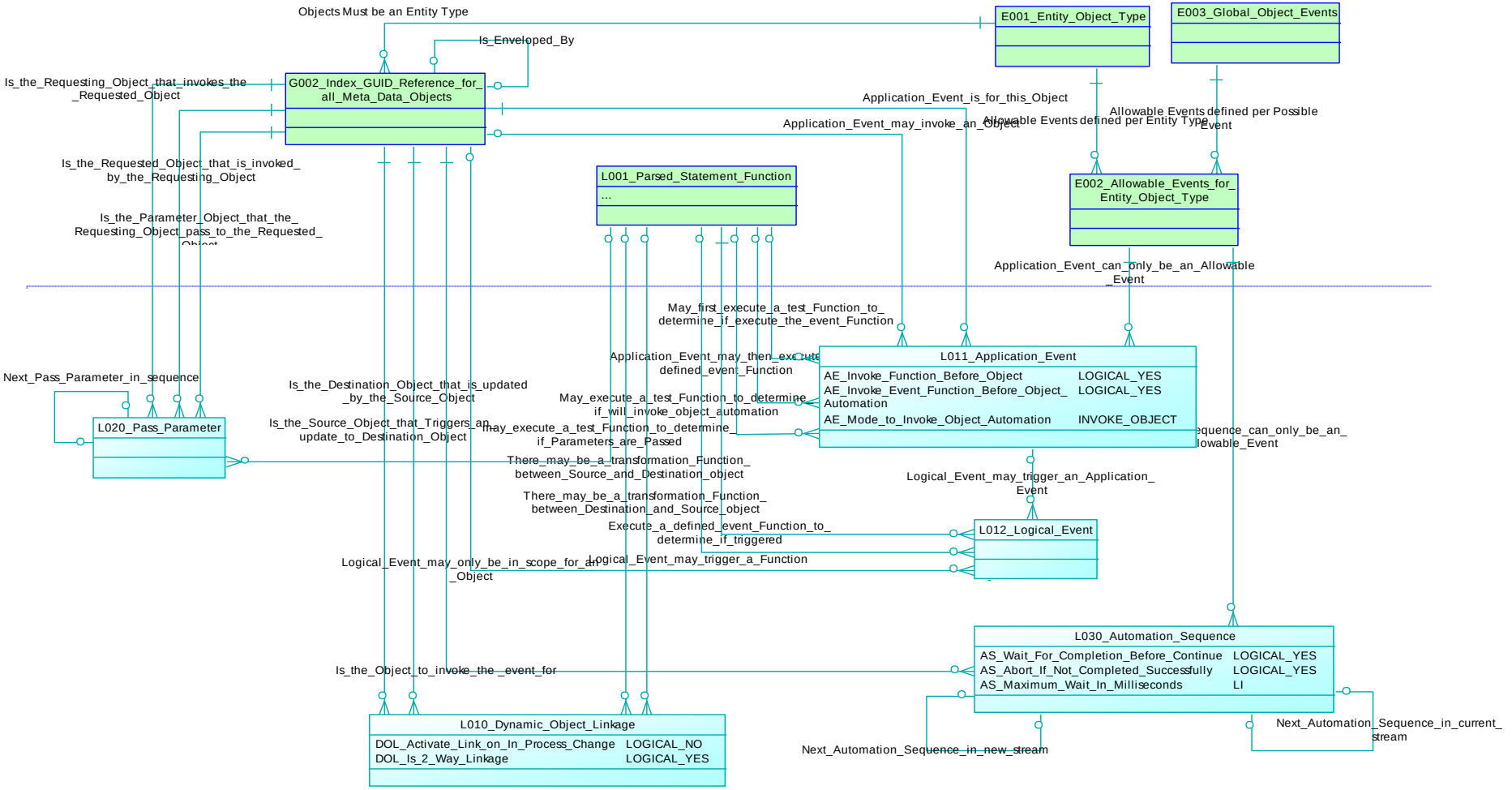


Figure 30 – Event Processing Model

The visual structure events uses the following entities to model the definition of the events. These entities are primarily general reference entities:

- **Entity Object Type**: is a reference list of the defined visual structure entities e.g. Canvas, Freeform Panel etc. These are listed as the row headings of Table 6.
- **Global Object Events**: is a reference list of the globally available object events. The visual structure objects from this list are listed as the column headings of Table 6.
- **Allowable Events for Entity Object Type**: are the available Global Object Events for each of the defined Entity Object Types. These are listed as the table contents of Table 6.
- **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) as classified by their Entity Object Type. This reference will identify the object whose event is to be selected.

These entities provide the primary ability to model the required application logic using the relationships of visual structure elements:

- **Dynamic Object Linkage**: allows any defined meta-data object to be defined with a dynamic linkage to any other meta-data object, allowing any updated value to be automatically updated to the other object. E.g. of particular use in displaying and interacting with data from data sourced controls.
- **Application Event**: in the visual structure context, is where any non-automatic visual structural event is defined as part of the required application logic by assigning specific functionality to a selected event for a selected visual object. The available actions are:
    - Can define an Event Function to be executed.
    - Can choose to fire the Event Function before (or after) any automation logic that occurs automatically as a result of the defined visual structure elements e.g. a Canvas will automatically invoke each of the Freeform and Navigation Panels defined for that Canvas in the defined order.

- Can also define a Test Function to be executed that will execute before an Event Function to determine if the Event Function should be executed. This allows for greater commonality of features to be provided in Functions and for local determination to be implemented or tested on an instance basis, if required.
- Can also define a Test Function to be executed that will execute before any automation logic to determine if the automation logic should be executed.
- Can choose to invoke a specific mode for any subsequent automation logic that occurs automatically as a result of the defined visual structure elements. See 5.3.3.4 Visual Element Function Call .

- **Parsed Statement Function**: are any specific defined Functions that are to be triggered for the Application Event.

The **Allowable Events for Entity Object Type** for the visual structure elements are listed in Table 5. The list is quite minimal as it only needs to support the use of the visual structure elements in the day to day operation of the meta-data EIS application by users, rather than the more extensive set of events that would be available in development environments.

Visual Structure Element Events	**Notes**
Start	When the object is first referenced by another reference (similar to constructor event in development environments).
Close	When the object is to be removed from operation or access (similar to destructor event in development environments).
Operate	When the object is clicked in normal operation of the features of that object. E.g. selecting the dropdown component of a Drop Down Box.

Visual Structure Element Events	Notes
Click	When the object is clicked but not in a location that would provide any expected result from the normal operation of the object. E.g. clicking an Image or an unused area on a Canvas doesn't normally invoke any functionality but could via this event.
UserUpdate	Whenever an object that can change its state or value via normal operation, does change its state or value. E.g. each time a user types a character in a Text Box.
AutoUpdate	Whenever a linked object is updated as a result of another operation occurring in the system, usually by another user of function.

Table 5 - List of Available Events for Visual Structure Elements

The matrix of allowable events for each visual structure element (extract of Allowable Events for Entity Object Type for visual structure elements) is listed in Table 6, where X denotes the event is available for that entity, and – denotes the event is not available.

Note the following rules for entities such as Navigation Panel which defines the primary object (the parent) and Navigation Panel used on Canvas which defines a usage instance (the child):

- The parent can disallow any child events to be defined.
- Events defined for the parent object definition can be forced to override any child events.
- Events defined for the parent object definition can be forced to override only undefined child events.
- Depending on the parent rules, the child can chose whether to inherit any parent events.
- Depending on the parent rules, the child can chose to inherit parent events only if there are no child events defined at all.

Allowable Events / **Visual Structure Elements**	**Start**	**Close**	**Operate**	**Click**	**UserUpdate**	**AutoUpdate**
Canvas	X	X	-	X	-	-
Navigation Panel (and **used on Canvas**)	X	X	-	X	-	-
Navigation Panel Item	X	X	X	X	-	-
Freeform Panel (and **used on Canvas**)	X	X	-	X	-	-
UI Line, Rectangle, Ellipse, Text, Image, Video (and **used on Canvas**)	X	X	-	X	-	-
UI Button (and **used on Canvas**)	X	X	X	-	-	-
UI Data Grid, Text Box, Selection (and **used on Canvas**)	X	X	X	X	X	X
UI Tree (and **used on Canvas**)	X	X	X	X	-	X
UI Tab (and **used on Canvas**)	X	X	X	X	-	-
UI Cross Tab (and **used on Canvas**)	X	X	-	X	-	X
UI Slider (and **used on Canvas**)	X	X	X	-	X	X

Table 6 - List of Allowable Events for each Visual Structure Element

5.3.3.4 Visual Element Function Call

As discussed in 5.3.3.1 Automatic UI Generation the visual structure elements will automatically generate aspects of the application based on the defined visual structure relationships.

The Visual Element Function Call allows the definition of manual program flow to visual structure elements by allowing any function to invoke a visual structure element as part of its definition

This feature is implemented as the VISUAL() function which allows any defined visual structure element to be called directly for execution. The VISUAL() function is invoked with the following arguments:

- **Type**: the meta-data entity type e.g. Canvas, Freeform Panel Used On Canvas etc.

- **Name**: the name (of that type) of the object that is to be invoked e.g. ‘Employee Names’ may the name of a Freeform Panel Used On Canvas that is used to manage the parts of an employee’s names.
- Any number of defined Events may be defined in the function call, as lists of (Event, Mode, Function):
    - **Event**: which of the allowable events for that object will be overridden.
    - **Mode**: how the mode of that event may affect the normal defined functional and structural execution of the object for this invocation:
        - **Cancel**: do not execute any of the normal defined functional and structural execution of the object.
        - **Execute**: execute the provided function as a replacement of the objects Start or Close event function code.
        - **ExecuteNV**: execute the provided function as a replacement of the objects Start or Close event function code but in a non-visual mode where none of the structural elements are displayed for user interactivity.
        - **Function**: execute only the defined functions, but not any automatic structural components, of the object.
        - **Normal**: execute normally as per the current definition of the object.
        - **StructAll**: execute only the automatic structural components, and of any component structures, but not any defined functions, of the object.
        - **StructOnly**: execute only the automatic structural components, but not invoke any component structures, nor any defined functions, of the object.
    - **Function**: an optional function command that may be executed as an alternative to the object’s defined event function.

This function allows the defined visual structure elements to execute with modified behaviour for any number of altered circumstances or usage, including in a non-visual processing mode.

5.4 Program Flow Elements

A significant proportion of the meta-data EIS application can be readily and simply modelled and defined as visual structure element components and indeed, when coupled with the basic definitions of the data sources (see), may be adequate for many simpler data applications – which could be based entirely on easily defined options in a meta-data EIS application meta-data editor, with no need for functions or additional logic.

Currently however, any realistic meta-data EIS application will require a greater level of sophistication than would be offered in the first generations of meta-data EIS application meta-data editors, so additional functionality may need to be defined which are implemented in the meta-data EIS application as user-definable functions, with a complexity not unlike that already commonly experienced in popular products such as Microsoft Excel.

As the options, templates and sophistication of meta-data EIS application meta-data editors increases, they will be expected to also offer increasingly sophisticated template options, which could include an ever expanding set of built-in functions and selectable options that could conceivably drastically reduce even the need for user-defined functions to a low level.

In addition to modelling the visual structure elements and function processing capability into the meta-data EIS application model, I have also included additional capabilities that can provide significant and novel application functions – for Application Workflow and Variant Logic. These features can be accessed without any coding nor the requirement to integrate with a third party application.

The meta-data EIS application model supports the following program flow elements:

- **Visual Structure Generation and Access**: defines the basic automatic program flow due to the visual structure elements.
- **Visual Structure Element Events**: are defined and available for each of the defined visual element types and can be defined to invoke other visual elements and/or call functions to execute.
- **Functions**: high level feature sets exposed as functions to implement features normally defined in code to allow for the non-visual processing requirements of application modelling.

- **Application Workflow**: provides a more targeted focus on defining the steps that are required to achieve a required outcome and specifying the appropriate logic that will support those requirements.
- **Variant Logic**: the capability for users to alter the pre-defined application logic to their own definition.

Each concept is discussed in more detail in the following sections.

5.4.1 Visual Structure Generation and Access

This key aspect of the meta-data EIS application, that the defined visual structure elements will be automatically instantiated by the runtime engine upon model execution and thus provide the major source definition of the initial and ongoing visual application logic is discussed in detail in 5.3.3.1 Automatic UI Generation .

Additionally, the visual structure elements can be accessed as a Function. The Visual Element Function Call allows the definition of manual program flow to visual structure elements by allowing any function to invoke a visual structure element as part of its definition. See 5.4.3 Functions for meta-data EIS application functions.

This feature is implemented as the VISUAL() function which allows any defined visual structure element to be called directly for execution. The VISUAL() function is invoked with the following arguments:

- **Type**: the meta-data entity type e.g. Canvas, Freeform Panel Used On Canvas etc.
- **Name**: the name (of that type) of the object that is to be invoked e.g. ‘Employee Names’ may the name of a Freeform Panel Used On Canvas that is used to manage the parts of an employee’s names.
- Any number of defined Events may be defined in the function call, as lists of (Event, Mode, Function):
    - **Event**: which of the allowable events for that object will be overridden.
    - **Mode**: how the mode of that event may affect the normal defined functional and structural execution of the object for this invocation:
        - **Cancel**: do not execute any of the normal defined functional and structural execution of the object.

        - **Execute**: execute the provided function as a replacement of the objects Start or Close event function code.
        - **ExecuteNV**: execute the provided function as a replacement of the objects Start or Close event function code but in a non-visual mode where none of the structural elements are displayed for user interactivity.
        - **Function**: execute only the defined functions, but not any automatic structural components, of the object.
        - **Normal**: execute normally as per the current definition of the object.
        - **StructAll**: execute only the automatic structural components, and of any component structures, but not any defined functions, of the object.
        - **StructOnly**: execute only the automatic structural components, but not invoke any component structures, nor any defined functions, of the object.
    - **Function**: an optional function command that may be executed as an alternative to the object's defined event function.

This function allows the defined visual structure elements to execute with modified behaviour for any number of altered circumstances or usage, including in a non-visual processing mode.

5.4.2 Visual Structure Element Events

A limited set of events are defined and available for each of the defined visual element types and can be defined to invoke other visual elements and/or call functions to execute. These are discussed in detail in 5.3.3.3 Visual Structure Element Events .

5.4.3 Functions

One of the guiding principles of the meta-data EIS application model is that EIS applications tend to have common features (such as data entry into data based forms) that can progressively be modelled into high level model elements that can provide the required features based on the re-use of these high level model elements with different sets of instance data or definitions.

The visual structure elements are the result of refining the visual aspects of application modelling. The use of a high level feature set exposed as functions to replace features normally defined in code is the result of reviewing the non-visual requirements for application modelling.

While certain logical processing sequences may be implemented in the meta-data EIS application as complicated looking nested functions, I consider the use of functions as a reasonable and necessary alternative to code based on the following reasons:

- In general, functions and certainly the meta-data EIS application function definitions have a much simpler definition than is typically required for common modern language based classes and methods which would be incomprehensible to the average business user.
- Common business products such as Microsoft Excel and its available functions are regularly used by business users throughout the world, and to great sophistication by power users of these products – there is already a widespread business user base familiar with nested function logic.
- Functions, whether smaller single line, or more complicated multi nesting, are defined and stored on an individual basis in the meta-data EIS application model, allowing for individual configuration management that can offer atomic and automatic change level version management, control and deployment. In code based systems, logic is usually grouped into larger compiled modules that most typically require static deployment, and rarely only a higher level of dynamic deployment at best.
- The atomic nature of functions in the meta-data EIS application model with their clear identification and complete separation from any other logic or visual components allows greater opportunity for any execution optimisation vs the typically combined and embedded code based class and method definitions. This is potentially an important optimisation consideration for meta-data EIS applications.
- The emerging usage of smartphones and tablets have developed such highly used apps as IFTTT (IF This Then That) where users are presented with simple objects and events from apps on their device and are able to readily define simple logic steps to help them manage the often high

volumes of communications and social media messages. Non-technical users have embraced simpler programming metaphors.

Figure 31 provides an extract of the conceptual design for the function model design. This design allows for the nested functions to be modelled to an internal atomic level which is required for model validation. However a runtime execution engine is not limited to reconstruction of a potentially nested function at each invocation and due to the logical separation of functions from all other logic, runtime execution engines can choose to atomically manage and execute alternate format optimisations of functions where they may provide expected performance enhancements.

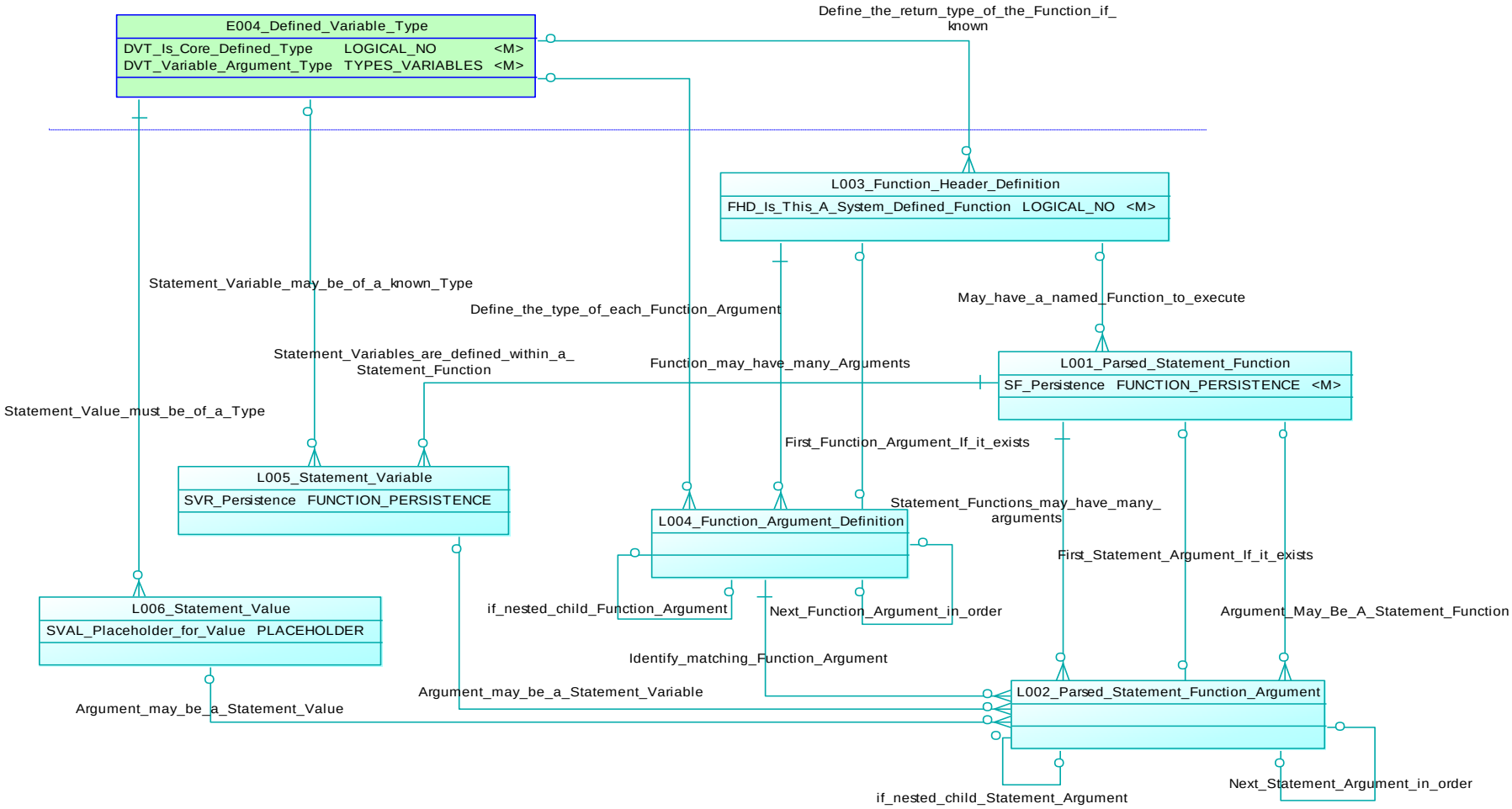


Figure 31 – Functions Model

The functions design uses the following entities to model the definition of the function logic. These entities are primarily general reference entities:

- **Defined Variable Type**: is the list of the types that can be defined for variables and data items. The types can be extended upon to define additional user defined types.
- **Statement Variables**: is the list of defined variables that have been specified in any Parsed Statement Function.
- **Statement Values**: is the list of defined values that have been specified in any Parsed Statement Function.

These entities provide the primary ability to model the required application logic using the relationships of visual structure elements:

- **Function Header Definition**: the definition and identification of any named functions including the core system defined functions e.g. AVERAGE(), RECORD() as well as any vendor or user defined functions.
- **Function Argument Definition**: are the individual argument definitions for the named functions. Functions may have any number of comma delimited defined arguments including nested arguments (which also need to be individually defined). Nested arguments are grouped within braces () as a list of comma delimited arguments.
- **Parsed Statement Function**: are the parsed and stored structure of any specific instance of the use of defined Functions. They are always an instance of a predefined Function.
- **Parsed Statement Function Argument**: are the individual argument definitions for each instance of a Parsed Statement Function. Each argument may be:
    - A Statement Variable,
    - A Statement Value,
    - A Parsed Statement Function, or
    - A nested argument list.

A summary of the classes of functions defined for use for meta-data EIS applications matrix is listed in Table 7. A full list of functions, syntax and usage notes

is detailed in Chapter 8 - Universal Access to Temporal Meta-Data Framework for EIS in the Cloud.

Function Classifications	**Purpose**
Processing	Define, modify and manage internal meta-data. Execute visual and non-visual components of the meta-data logic processing.
Database	Whenever a linked object is updated as a result of another operation occurring in the system, usually by another user of function.
Logical	Perform logical testing and conditional processing.
Group Data	Perform a basic analysis and searching of data from a View Table. Provide basic statistical calculations.
Date / Time	Convert to and from date / time formats. Decode date / time data into common representation formats.
Mathematical	Convert to and from integer and numeric formats. Provide fundamental scientific calculations.
Text	Convert to and from text and character based formats. Provide text search, extraction and replacement.

Table 7 - List of Function Classifications

5.4.4 Application Workflow

The logical workflow of an EIS application is the combination of all of the visual elements of the application combined with the underlying programmed options that support the visual and non-visual aspects of the application. E.g. click on the button labelled Save and the updated data on that user screen will be submitted as a transaction to update and save the data. The meta-data EIS application seeks to provide an identical user experience throughout the application although the underlying logical workflow has been based on a defined, stored and executed model rather of disparate code segments.

The application workflow of an EIS application is a more targeted focus on defining the steps that are required to achieve a required outcome and specifying the

appropriate logic that will support those requirements. The common workflow options implemented in EIS applications are:

- **None**: no offered workflow capability or external integration.
- **Wizards**: additional usually sequential user interface screens that prompt and guide the user through the specific programmed interaction – these wizards may be the primary user interaction method or may be in addition to other user entry screens.
- **Internal**: additional user selectable or definable workflow options. Usually providing support for defining the workflow sequence and addressing existing user screens and content. Many options require coding the workflow solution using the EIS applications API or SDK.
- **Third Party**: numerous standalone workflow management systems exist that specialise in defining sophisticated workflow logic but require access to EIS applications functions for integration – this can be achieved if the EIS application is compatible with the third party workflow system or to an acknowledged workflow business standard, or may need to be coded for each EIS application integration through its API or SDK..
- **Cloud Access**: where an EIS application exposes its internal functions to a globally accessible standard, such as web services, aspects of the EIS application can be securely managed as workflow components of other EIS and workflow applications.

The concept of application workflow is of key importance as it directly supports the capability of the EIS application in achieving the aim of an ever closer integration with the business objectives of using the EIS application. Providing the appropriate functionality to users, as can be modelled and provided by the meta-data EIS application, cannot always be fully exploited unless the subsequent application usage is most directly targeted to the appropriate users for; access, processing and authorisation.

The meta-data EIS application provides extensive support for defining application workflows:

- **Content**: user screens can be defined as the common static structure / dynamic content model of typical EIS applications. All screen components can be readily re-assembled and reused to create specific

wizard workflow sequences without coding. This feature is a standard aspect of the model as described elsewhere in this chapter.

- **Access**: managing which users have appropriate access to data, transactions and authorisations is crucial to business operations. This is addressed in 5.6 Secure Access and Authorisation.
- **Processing**: providing the means to specify interaction sequences that apply to the transmission of information and any subsequent modification, updates, rejections etc, is described in this section.
- **Authorisation**: allowing the combinations of authorisations that can apply to the information, and the authorisation or rejection criteria, is described in this section.
- **External**: secure access to any of the meta-data EIS application components is provided to any external EIS or workflow application via the meta-data EIS application web service interface. This is described in Chapter 8 - Universal Access to Temporal Meta-Data Framework for EIS in the Cloud.

Figure 32 provides an extract of the conceptual design for the application workflow model design. This design allows for multiple branching workflow stages to be modelled, based on user events or data state analysis, with different authorisation states and authoriser groups and combination can be defined for each workflow stage. The workflow also supports multiple alternate success states at different stages.

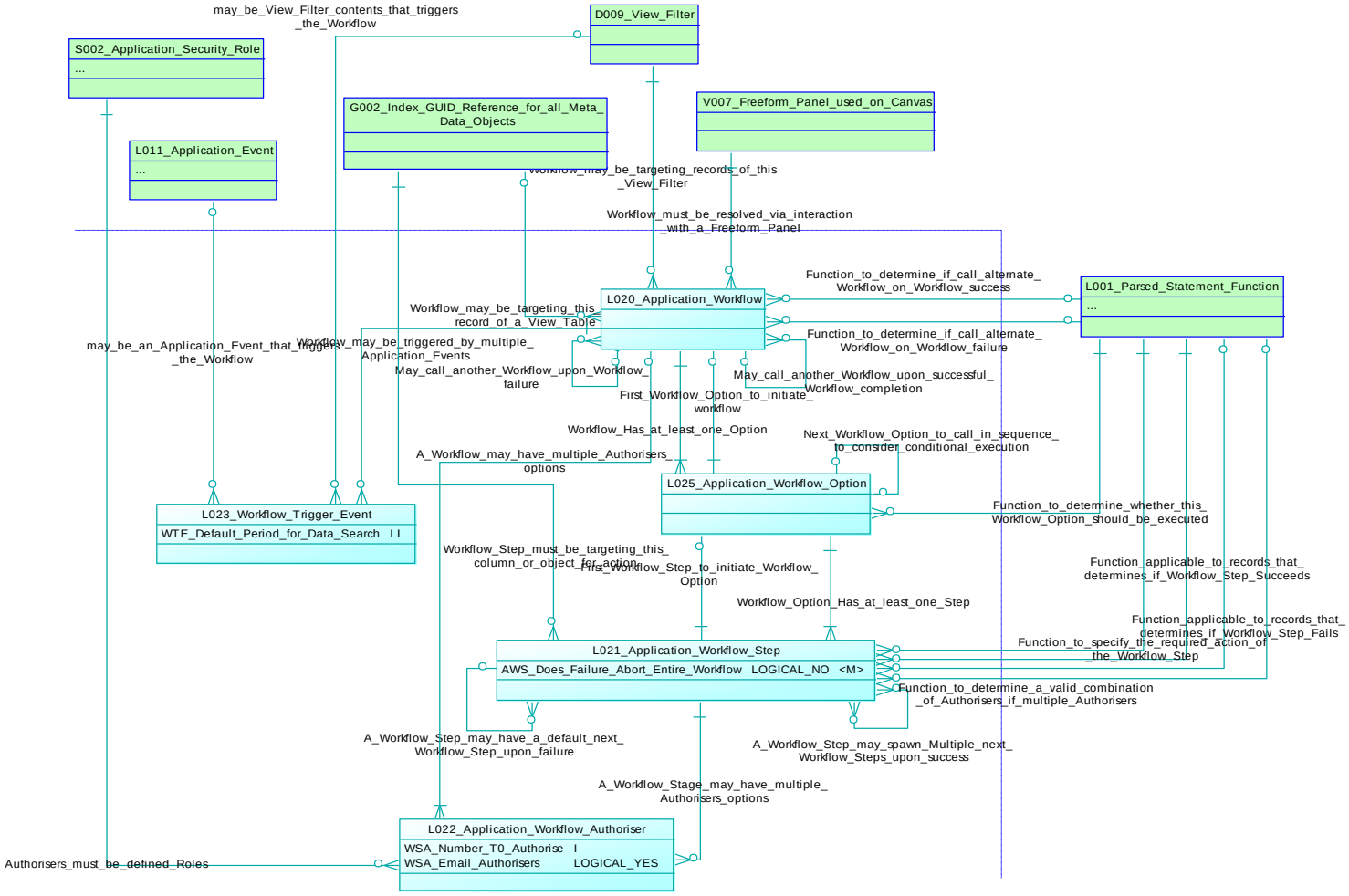


Figure 32 – Application Workflow Model

The application workflow design uses the following entities to model the definition of the workflow logic. These entities are primarily general reference entities:

- **Application Security Role**: is the list of the available application roles that can be specified as authorisers for a workflow stage.
- **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) as classified by their Entity Object Type. This reference will identify the object that is the target of the workflow.
- **View Filter**: is a defined data filter for a View Table. A View Filter may be the definition of the source data records as the subject of the workflow.
- **Freeform Panel used on Canvas**: is used to share Freeform Panel objects between Canvas objects. This will define the user interface object that will be used to view, amend or authorise the workflow.

These entities provide the primary ability to model the required workflow logic:

- **Application Workflow**: the high level definition and identification of the workflow. The workflow is considered to succeed when one of its workflow stages succeeds that does not have any further workflow stages i.e. success at the end of a workflow stage chain. The workflow specifies:
    - the View Filter records that are the subject of the workflow,
    - the user interface Freeform Panel that will be used to resolve the workflow,
    - if any alternate workflows will be called upon this workflow succeeding or failing, and any functions that will be used to determine if these alternate workflows will be called,
    - the first workflow stage used to commence the workflow processing.
- **Application Workflow Stage**: are the individual branches or stages that a workflow may be required to proceed through in order to succeed. Multiple parallel stages for a particular workflow may be undergoing independent authorisation concurrently. It specifies:
    - The specific object or data column that is the subject of this workflow stage,

    - functions that will be used to determine if the workflow stage succeeds or fails,
    - function to specify a successful combinations of authorisers when multiple groups of authorisers can be selected,
    - if any alternate workflow stages will be called upon this workflow succeeding or failing,
    - if failure of this workflow stage aborts the entire workflow.
- **Parsed Statement Function**: are the parsed and stored structure of any specific instance of the use of defined Functions. They are always an instance of a predefined Function for the listed workflow purpose.
- **Application Event**: in the visual structure context, is where any non-automatic visual structural event is defined as part of the required application logic by assigning specific functionality to a selected event for a selected visual object. This may be the manual user event that triggers the workflow.
- **Workflow Trigger Event**: are the selected events that will trigger the workflow. Eligible events are:
    - Application Events,
    - Record sets as defined by a View Filter. The data search period can be defined for how frequently the View Filter query is executed to determine any eligible records for this workflow.
- **Application Workflow Authoriser**: are the Application Security Roles that identify the potential authorisers for the workflow and/or stage. For simple authorisations where only a single group of authorisers is specified, the number of required authorisation for success can be specified. Where multiple groups of authorisers are specified, different combinations of successful authorisers can be specified as a function for the workflow and/or stage.

The application workflows of the meta-data EIS application can work as an additional super-layer of the general application logic, by reusing existing meta-data EIS application components in both wizards and in workflow authorisations. The further benefit of the meta-data EIS application is that this all can be achieved without coding, enabling business users to define their own wizards and workflow sequences

to further optimise their own business roles, in addition to workflows that are provided by the original vendor.

The secure access to all meta-data EIS application components from external cloud based EIS or workflow applications via web services further expands on the level of global extensibility available by meta-data EIS applications.

5.4.5 Variant Logic

Almost every application that is in practical business use is the result of hard coded program logic that has been compiled and/or deployed for use as part of a developer's release schedule.

The scope for end users to influence the design and functionality of the application is usually minimal and limited to providing suggestions or advice to the developers although some senior management in large corporations may have the opportunity for closer consultation.

While identified bugs in an application may be a priority and receive a higher level of attention in terms of feedback from users and the response of vendors, it is more typical that user requests for change may have long periods before they are introduced into production application releases, if ever.

Expensive alternatives that are often employed by organisations are to engage the vendor or authorised third parties to develop specific customisations for the user requirements to become embedded within a new localised version of the application. Whilst it may have a suitable overall business case for an organisation to take this option, it is often expensive, and can cause additional delays and expense when the core application is upgraded or patched due to potential introduced incompatibilities between the new version and the user customisations.

We have seen how the meta-data EIS application is defined as a model, without coding, for execution by the runtime engine. The model definition can be created by a meta-data editor which provides a much simpler alternative to defining the application logic, compared to the IDEs for professional programmers, and thus these same model definition tools can be available to business users to define application logic for meta-data EIS applications.

In addition to business users defining and creating their own application logic, another key aspect of the meta-data EIS application is the capability to alter the pre-

defined application logic as what I term Variant Logic, to become a variation of the application logic for a modified purpose.

Variant Logic can be applied to any object defined in a meta-data EIS application. It can be defined by any user and can be executed by any user as an alternative to the standard application logic. Of course, there are available security options that can be applied to manage all of these options, particularly where there is important application logic that must be adhered to.

Users of business applications would be familiar with some levels of application configuration and customisation provided in some applications, such as:

- The ability for users to select their own colour scheme for aspects of the user screens,
- The option to set some environmental options e.g. international locale to adjust some items' display,
- Save the screen positions of user positioned screens,
- Create simple reports or user defined data extractions and save them as accessible objects.

The meta-data EIS application provides for such simple examples and much further by allowing every defined component of the meta-data EIS application can be changed to become Variant Logic. Some examples of how users can adjust a meta-data EIS application to more closely suit their local processes are:

- **Application example**: Payroll clerk may modify the initial starting Canvas of a financial application so that instead of the same application start-up screen, their more regularly used Payroll Processing Canvas is always displayed first.
- **Canvas example**: when performing reviews of staff's timesheet entries on their Timesheets screen, the Payroll Supervisor may choose to reorder the positioning and change the sizing of various Freeform Panels on the screen to say; as they have a very large monitor, they greatly increase the size of the Freeform Panel which displays the tabular data to expand and show more columns together, as well as moving the individual row data Freeform Panel from below the tabular data Freeform Panel to below.
- **Navigation Panel example**: the Finance Manager has created several reports that are regularly reviewed, as well as some new screens that they

have defined themselves to help with workflow approvals for overtime payments. The common Navigation Panel that they use is updated with new objects to include these new GUI Reports and Canvases. The General Manager also finds these very useful, so also now accesses the changed Navigation Panel.

- **Freeform Panel example**: the financial system was procured from a vendor in the United States and includes by default references to an employee's 401K retirement savings. The Finance Manager has altered the names of several 401K related objects to more relevant local superannuation terms as well as changed the supporting text and help files. There are also some data columns that are not relevant at all to local conditions so these have been removed completely from all Freeform Panels and from the View Tables that are typically used. These changes have been assigned to all users of the finance application to ensure conformity.
- **Freeform Panel example**: a Personnel Officer continually uses the New Employee screen to enter new employees centrally for a large organisation. The default entry for each employee's Home Base is a free text entry, however there are usually only a few options so the Personnel Officer has changed that UI Object to become a GUI Selection and defined a short list of items with the most common as the initial default entry.
- **Freeform Panel example**: a Data Entry clerk has to transcribe the contents of hundreds of timesheets that are faxed from remote offices. The format of the timesheets no longer matches the order of entry on the Freeform Panel and some additional information is now required. The UI Objects on the Freeform Panel have been re-arranged to better suit the manual procedure, including resizing the objects to their most common entry size. New columns have been added to record the latest information requirements, and updated validation functions have been defined to help minimise the occurrence of data entry mis-keying. These changes have been applied to all other clerks entering the remote timesheets.

- **Function example**: a new category of employee payment was defined to allow for payments to be made under a federal parental leave scheme. The Payroll Supervisor defined the new data columns to track these hours and calculate the payable amounts, then updated the calculation collation function to include the new amounts for each person's payroll.
- **Workflow example**: the original workflow to authorise overtime payments needs to be changed. There is a new role in the organisation called Divisional Manager that needs to approve any record where the overtime hours are more than 50% of an employee's normal hours. A new workflow stage was inserted to achieve this.

By extending the above simplistic examples with more complexity, including adding entirely new functionality, the accessibility, power and immediacy of the Variant Logic becomes a key capability of the meta-data EIS application.

The meta-data EIS application provides the following aspects to implement Variant Logic:

- **Logic Definer Access**: is a secure process that defines the levels of object change authorisation, and the roles of authorised Logic Definers (and those who can perform the role) who perform the actual logic changes.
- **Variant Access**: once Logic Definers have defined the application logic changes, the Variant Logic becomes available for access. Access can be assigned; application wide for all users, based on roles, or for individual users.

These aspects are further discussed in the following sections.

5.4.5.1 Logic Definer Access

The golden rule of application logic definition in the meta-data EIS application is that the original author, definer or owner of application objects or Variant Logic objects maintains the ownership of those objects and the ongoing authorisation to modify those objects. Such modifications will be managed through the standard temporal meta-data structure.

Similarly, new application logic can be defined through the creation of new meta-data objects, however the new logic elements can only be accessed by creating Variant Logic to existing objects, in order to then access the new meta-data objects. All such access is managed by an authorisation process.

The structure of the Logic Definer Access describes a secure process that defines the levels of object change authorisation, as a hierarchy of authorisation, and the roles of authorised Logic Definers within those levels (and those who can perform the role) who perform the actual logic changes. Any changes are conducted under an overall authorisation umbrella that determines what content the Logic Definers are able to change. Multiple individual meta-data changes can be grouped as a particular Variant Logic instance.

Figure 33 provides an extract of the conceptual design for the Logic Definer Access model design. This design allows for assigning hierarchical authorisations, defining the aspects of the meta-data definitions that can be modified, and assigning the change permissions based on these aspects.

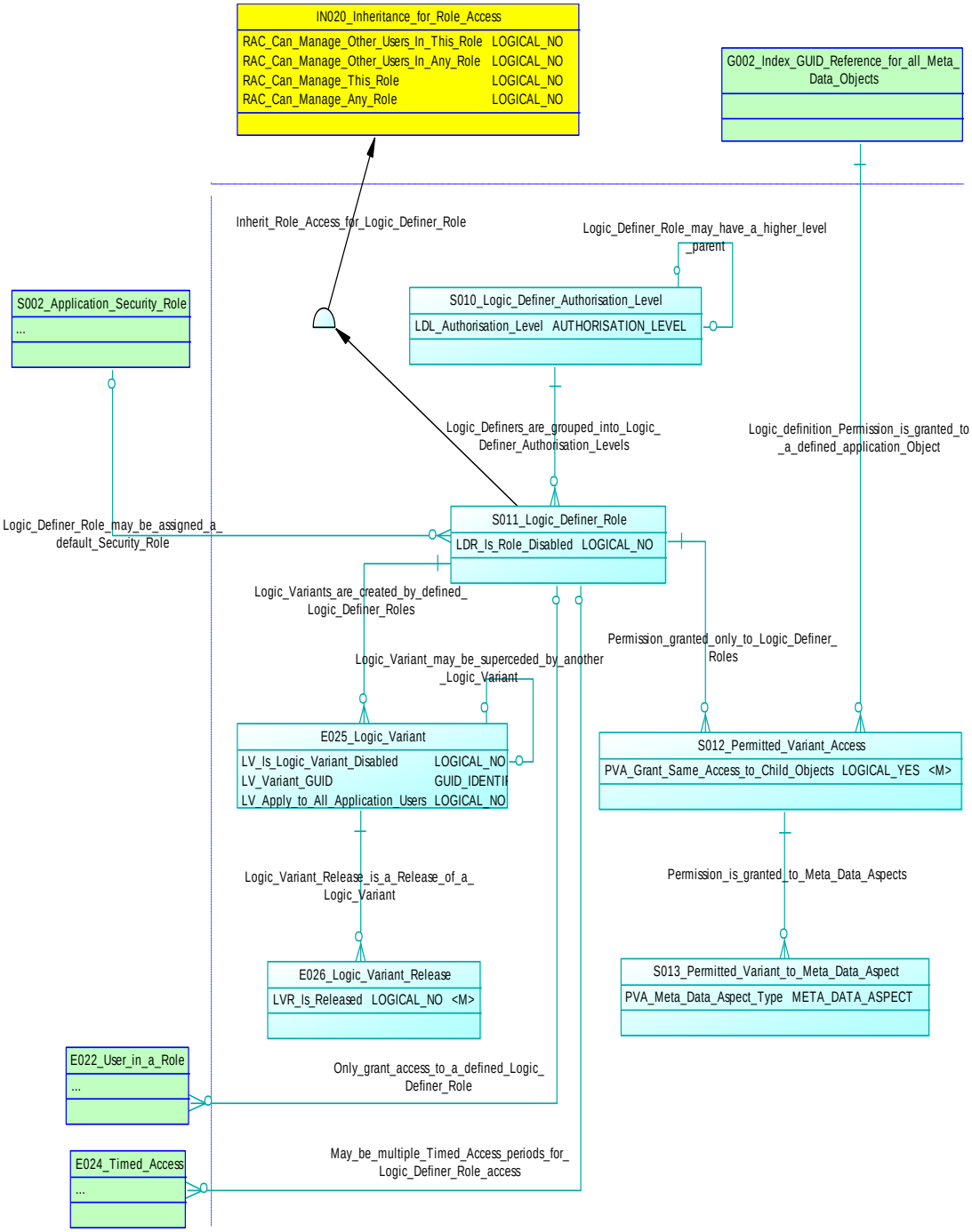


Figure 33 – Logic Definer Access Model

The Logic Definer Access design uses the following entities to model the definition of the change access. These entities are primarily general reference entities:

- **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) as classified by their Entity Object Type. This reference will identify the objects for determining the level of change access available.
- **Application Security Role**: is the list of the available application roles. A Logic Definer Role may be assigned a default Security Role to provide the appropriate access.
- **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 for various access types, in this instance, for when Logic Definer Roles can operate.

These entities provide the primary ability to make the variant logic changes by the Logic Definer Roles:

- **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. An example of the highest level of authorisation downwards to the lowest is as follows:
    - **Vendor**: representing the original vendor of the meta-data EIS application as the highest authorisation level,
    - **Third Party**: considering authorised third party vendors that may be engaged.
    - **Corporate**: the highest internal authority of the owning organisation.
    - **Business Unit**: regional or individual segments of the owning organisation.
    - **Section**: individual local functional groups of the owning organisation.
    - **Users**: individual power users of the owning organisation

- **Logic Definer Role**: are the individual groups or roles that can be assigned to designate an identified group of functional logic definers. A Logic Definer Role is always assigned to a **Logic Definer Authorisation Level** which defines its relative overall authorisation level over other Logic Definer roles. A Logic Definer Role may also be assigned to an Application Security Role to clarify the basic object access for the role.
- **Logic Variant**: is a designated identifier to group all of the logic changes together into a practical set. The best use of a Logic Variant would be to group the associated changes of a set of new functionality. A Logic Variant is always the responsibility of a designated Logic Definer Role.
- **Permitted Variant Access**: identifies the objects that the Logic Definer Role has access to change as variant logic. Permission may be assigned to follow through to all child objects of that object
- **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 aspects to major aspects. Examples of meta-data aspect changes are:
    - **Text**: allowing a variant to rename non-identifying textual characteristics.
    - **Colours**: allowing different colours to be assigned to objects and backgrounds.
    - **Help**: update a variant's version of the help information.
    - **Sizes**: change the default display sizes of objects.
    - **Position**: move objects to different locations.
    - **Alignment**: adjust the alignment rules between visual objects.
    - **Access**: modify whether an object is accessed.
    - **Type**: modify the type of an object or major attribute.
    - **Function**: modify the function definitions that an object may use.
    - **Assign**: change any relationship assignments that are available for an object.
    - **Validation**: modify any validation rules.

The Logic Definer Access process offers a powerful capability to the meta-data EIS application by empowering authorised groups of users to configure and customise the meta-data EIS application to any permitted degree, without coding, and with the greatly reduced availability times and expense that the meta-data approach provides.

5.4.5.2 Variant Access

Once Logic Definers have defined the application logic changes, the Variant Logic becomes available for access, however the ongoing default access is to the original meta-data application logic. Access to any defined Variant Logic must be specified and can be assigned in several ways;

- **Application**: the changes are to be applied as the standard access for all users of the application,
- **Security Roles**: the changes will apply to all users that belong to the specified Security Role.
- **Security User**: the changes will apply to individual users.

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 (see 5.6 Secure Access and Authorisation).

Figure 34 provides an extract of the conceptual design for the Variant Access model design. This design allows for assigning access to; users, roles or an application wide basis.

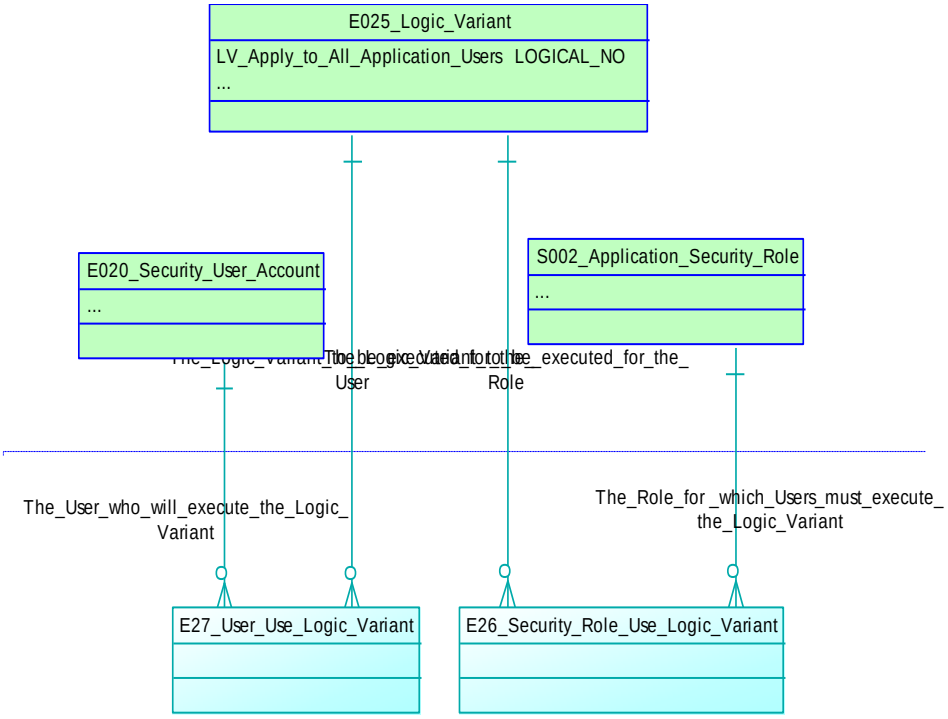


Figure 34 – Variant Access Model

The Variant Access design uses the following entities to model the definition of the alternate logic access. These entities are primarily general reference entities:

- **Logic Variant**: is a designated identifier to group all of the logic changes together into a practical set. Has the option to designate whether the Logic Variant is to apply to all users of the application.
- **Security User Account**: is the list of Users that are defined in the application runtime execution environment. These users may be granted individual access.
- **Application Security Role**: is the list of the available application roles. All users assigned to a Role may be granted access.

These entities provide the primary ability to assign the variant access to the Variant Logic:

- **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.

The Variant Access process is a simple mechanism to assign the ongoing runtime access of users to the new Variant Logic as an alternative to the existing original logic and other available logic options..

5.5 Extended Data Dictionary

The ability to access data in a simple and meaningful way is a key aspect of the meta-data EIS application. Data in the real world is often named and stored using incomprehensible formats which contribute greatly to misunderstandings and misuse. The meta-data EIS application seeks to assist with providing abstraction layers around the database access that permits a simplified and more relevant presentation of the data to the users of the meta-data EIS application.

A key expectation is that the ultimate storage and database management system will be SQL based, although this will be dependent on the interfaces supported by the runtime engine which may provide access to other storage structures. From the perspective of the meta-data EIS application, the starting place is that there is a simple model of database tables available from which to build upon to define the more user friendly modelling abstractions that are described within this section.

The key aspects of the meta-data EIS application data access are:

- Real data storage and access is managed by the runtime engine.
- The starting point for the meta-data EIS application is a virtual representation of the data in a tabular format with identified relationships and data types mapped to those of the meta-data EIS application. An initial abstraction is permitted here akin to the role of defining a corporate data dictionary for the source data.
- The meta-data EIS application starts with a core set of defined data types and allows new types to be defined with the ability to convert between the new derived and original type.
- Data abstraction commences by applying alternate naming and formats to virtual data columns to become View Columns which become the primary source for data within the meta-data EIS application. Key data integrity issues such as validation rules and input masks can also be defined.
- Any combination of View Columns can be combined to become a new View Table which become the primary reference for data within the meta-

data EIS application. The View Table can provide similar functionality to common updateable database views although with additional capability. The underlying relationships of the referenced Virtual Columns and Virtual Tables will automatically determine the final row composition of the View Table. The abstraction permits the user to concentrate on what they want to do to the data, as how the data is processed will be managed by the runtime engine.

- View Columns can also be further aliased and abstracted for additional usage.
- View Tables can be further defined with multiple modifiers such as; View Filters, View Sorts and View Groups which refine the use of the View Table for individual usage throughout the meta-data EIS application.
- Transactions that modify data in View Tables are mapped back to the source Virtual Tables and Virtual Columns for commitment via the runtime engine.

This data access scheme allows for multi-level abstraction that progressively meets the requirements of the different needs and knowledge of the accessors:

- **Database**: the real stored data is maintained in its original or current form, as understood and managed by the organisation's database administrators. It is presented to and with access supported by the meta-data EIS application runtime engine.
- **Virtual Table / Columns**: is the lowest level mapping from the meta-data EIS application model to the presentation of table / column data from the runtime engine. This can be considered as a Data Dictionary style of abstraction and could be expected to be managed by organisational Data Administrators.
- **View Column / Tables**: is the usable abstraction level mapping of the meta-data EIS application. Meta-data objects can only interact with these View Column and View Table objects, and associated View Filters, View Sorts and View Groups.
- **View Column Alias**: authorised users can define their own aliased View Column objects that can also be modified for name, type and format (subject to appropriate conversion rules) as well as create their own View

Tables, View Filters, View Sorts and View Groups, in addition to all of the other meta-data objects, in the definition of their own application logic.

Figure 35 provides an extract of the conceptual design for the Data Access model design. This design allows for a data abstraction and access scheme is both very flexible and user friendly as it allows the different levels of data users the ability to define their own interpretations of the data in terms that are the most meaningful for their role.

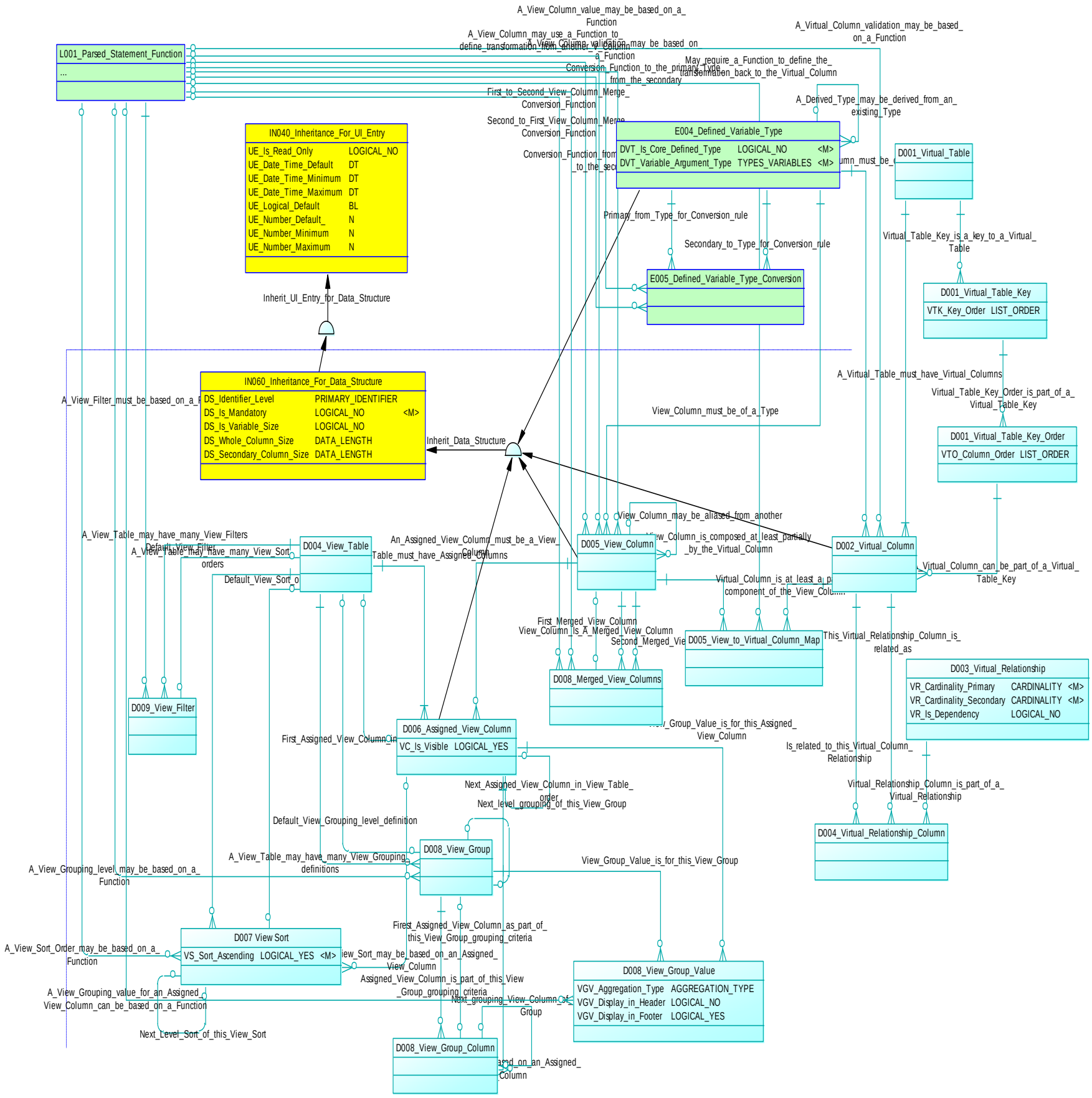


Figure 35 – Extended Data Dictionary Model

The Data Access design uses the following entities to model the definition of the data access. These entities are primarily general reference entities:

- **Parsed Statement Function**: are any specific defined Functions that may be used for; validation, data type conversion or for specifying table sorting, filtering or grouping rules.
- **Defined Variable Type**: is the list of the data types that are understood by the meta-data EIS application. Initially it consists of a core list of types such as; text, DateTime, Number, Logical, Object etc but can be extended with other user and system defined types by specifying additional derived types
- **Defined Variable Type Conversion**: is the additional information required for a derived type specifying the required conversion functions to translate between the derived and original type.

The Data Access type based entities will inherit common characteristics. As listed in Figure 35 these inherited characteristics include:

- **Inheritance For UI Entry**: basic object data entry features including; display mask, default values and ranges.
- **Inheritance For Data Structure**: specify the column characteristics such as; primary key, mandatory and sizing

These entities provide the primary ability to model the data abstractions for the meta-data EIS application data model:

- **Virtual Table**: Is part of the first level of data abstraction mapping to a real database table.
- **Virtual Column**: Is part of the first level of data abstraction mapping to a column of a real database table. The data type must map to a Defined Variable Type of the meta-data EIS application. A data validation function may be specified.
- **Virtual Relationship**: Is part of the first level of data abstraction recording the relationships and cardinality between Virtual Columns as identified in the source database structure.
- **View Column**: Is the key abstraction definition of data columns in the meta-data EIS application. Can apply alternate naming and formats to the source Virtual Columns to become View Columns which then become the

primary source for data for objects of the meta-data EIS application. The data type must map to a Defined Variable Type and additional data validation function may be specified. A View Column will either map to the source View Column or may be defined as an alias to another View Column to provide further user data abstraction.

- **View Table**: are defined as a collection of View Columns. The set of View Columns does not have to be from the same source Virtual Table. The set of View Columns will be defined as the Assigned View Column entity. The underlying relationships of the referenced Virtual Columns and Virtual Tables will automatically determine the final row composition of the View Table. View Tables can be further defined with multiple modifiers such as; View Filters, View Sorts and View Groups which refine the use of the View Table for individual usage throughout the meta-data EIS application.
- **Assigned View Column**: define the collection of View Columns that will constitute a Virtual Table.
- **Merged View Columns**: assist merging of multiple models by defining how semantically similar View Columns from originally disparate models can be directly associated to effectively merge the two View Columns automatically (see Figure 53 - Virtual Data Object Mapping Model Merging).
- **View Filter**: Acts as a set of selection or filter criteria applied to the View Table. The View Filter is how the nominal data set of a View Table can be limited to any specified subset for specific processing. A View Table may have a default View Filter. The View Filter is specified by a Function to determine the selection criteria.
- **View Sort**: provides a multi-level sorting capability for a View Table. A View Table may have a default View Sort. The View Sort is based on nested levels of sort criteria which can be based on either a Function or a View Column.
- **View Group**: provides a multi-level grouping capability for a View Table. A View Table may have a default View Group. The View Group is

based on nested levels of grouping criteria which can be based on either a Function or a View Column.

The Data Access abstraction provides a high level of abstraction by separating the meta-data EIS application object data requirements to use more understandable (to the user) data items. It also simplifies data transactions by avoiding any of the translations required to manipulate the source data which is managed via the abstracted model and the runtime engine. This allows the logic definers, whether vendors or users to concentrate on what they want to do with the data in a meta-data EIS application rather than how they need to manage the data.

Note that all meta-data in the model should also be accessed as pre-defined system View Tables and View Columns based on the final object names. This will allow direct access to the meta-data by meta-data based applications for meta-data management operations such as logic definer editors and security managers.

5.6 Secure Access and Authorisation

Access security is a fundamental requirement to ensure that only appropriately authorised actions and data transactions are enacted. The meta-data EIS application provides access security for two distinct areas:

- **Functional Access Security**: is the typical style of access security where users are assigned the features of the application that they can access, although the meta-data EIS application provides much finer grain control over the level of access definition as it can define access to the lowest level atomic object in the application.
- **Logic Definer Access**: is unique to the meta-data EIS application and provides security over which features of the meta-data EIS application can be modified by users. There is no parallel with the typically code developed EIS applications which can generally only offer minor configuration options whereas the meta-data EIS application is based on modelled objects which are defined by vendors and users, and may be modified by any authorised user.

The corresponding access security solutions are described in the following sections.

5.6.1 Logic Definer Access

The unique capability of allowing any authorised user to define new application logic or to modify existing application logic as new variant logic requires a separate authorisation structure. It is discussed in detail in 5.4.5 Variant Logic .

5.6.2 Functional Access Security

The general operation of the functional access security of the meta-data EIS application is similar to that of standard EIS applications in that the security is based on defined application roles and access to functions is granted to roles. A key distinction of the meta-data EIS application is that every single component object of the application can be subjected to access security vs the much broader security granularity offered in standard EIS applications.

Also, where standard EIS applications are typically static in their definition of what functions can be applied security access, the meta-data EIS application is truly dynamic as new application logic can be generated at any time by authorised users which becomes art of the overall application logic pool of objects for security access. However, as the meta-data EIS application access security is applied to its objects as they exist at any point in time, and can be applied on the basis of the relationship of its objects, the management of the dynamic object population can be efficiently managed via the same ongoing means, independent of application logic changes.

Figure 36 provides an extract of the conceptual design for the Functional Security Access model design based on role based access to the atomic objects of the meta-data EIS application model.

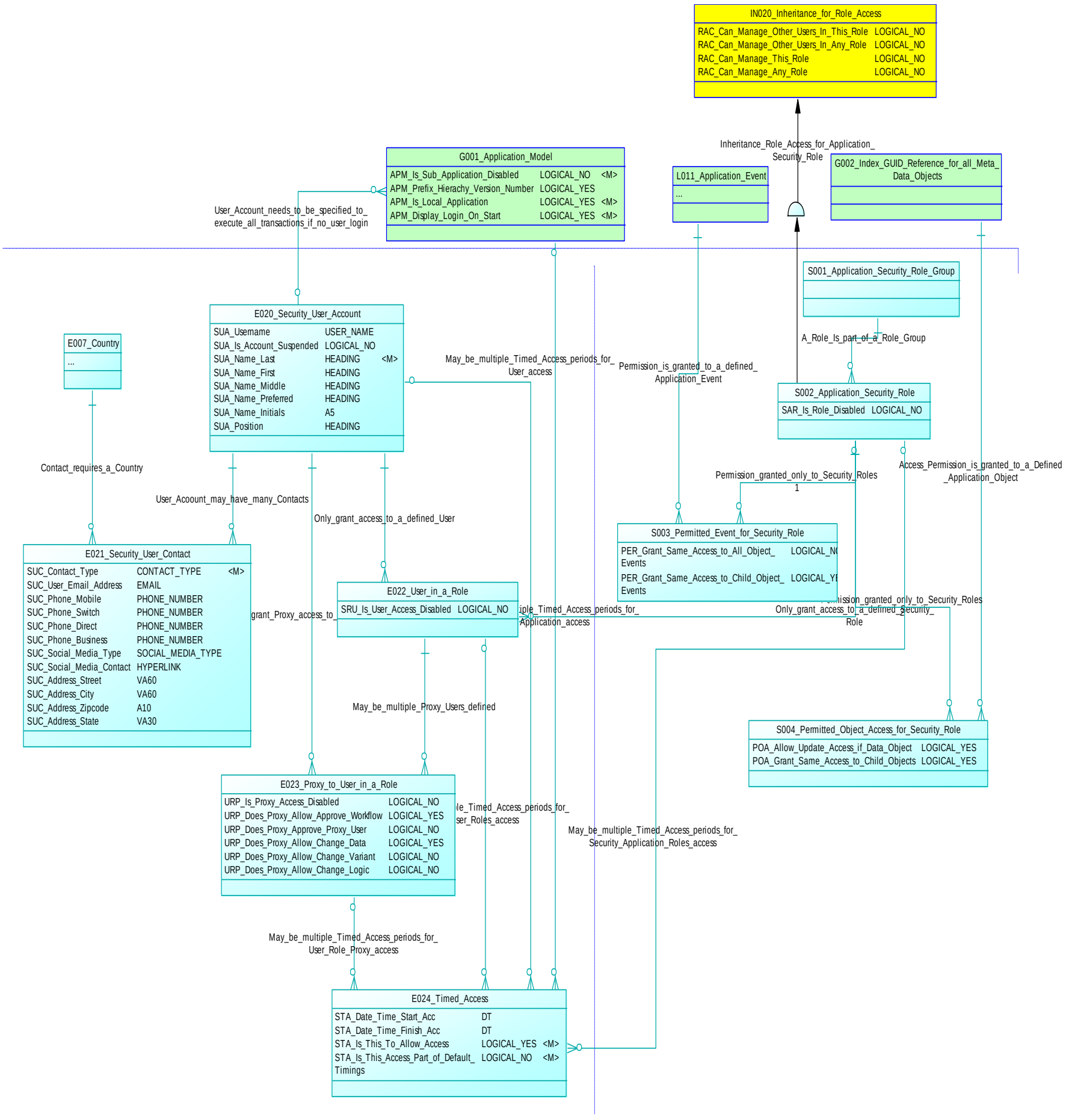


Figure 36 – Functional Security Access Model

The Functional Security Access design uses the following entities to model the definition of the secure access. These entities are primarily general reference entities:

- **Application Model**: is the high level identifier of the application as modelled in the meta-data EIS application. This model identifier is usually inherited throughout all model entities. Access to the entire application can be disabled, as well as defining timed periods for permitting or denying access.
- **Application Event**: in the general context, is where any non-automatic event is defined as part of the required application logic by assigning specific functionality to a selected event for a selected object. This includes the events of visual objects and of non-visual objects such as; functions, workflow and data access.
- **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) as classified by their Entity Object Type. This reference will identify the objects for determining the security access.

These entities provide the primary ability to define the security access to the application objects:

- **Application Security Role Group**: simply allows a grouping of the defined Application Security Roles for organisation.
- **Application Security Role**: is the list of the available application roles. Security access is based on roles rather than individuals. Access for all users allocated to a role can be disabled, as well as defining timed periods for permitting or denying access to all users allocated to a role.
- **Permitted Event for Security Role**: identifies the Application Event that the Security Role has access to. The basic assignation of access to the event allows execution of that event by all users allocated to the role. Options are provided to allow the same access to all events of that object, and to the events of child objects.
- **Permitted Object Access for Security Role**: identifies the application objects that the Security Role has access to. The basic assignation of access to the event allows read and display access to that object by all

users allocated to the role. Options are provided to allow update access if the object is a data source object, and access to all child objects.

These entities provide the primary ability to define the security access infrastructure to support the security objects:

- **Country**: is a list of countries used for localisation options. Used to identify the country of a user contact.
- **Security User Account**: is the list of Users that are defined in the application runtime execution environment. Users may be disabled from any application access, as well as defining timed periods for permitting or denying access to the user, granted access to applications via roles, or assigned as a proxy to another user.
- **Security User Contact**: provides the option of multiple location and contact details for users.
- **User in a Role**: is the list where Users are assigned to Application Security Roles and/or Logic Definer Roles. Roles may be disabled from any application access, as well as defining timed periods for permitting or denying access to the user.
- **Proxy to User in a Role**: is where Users may be assigned as a proxy to another user. Proxy users gain the ability to operate as their proxied user. The proxy role may be disabled, as well as defining timed periods for permitting or denying proxy access. Limitations can be applied to the proxy access such as;
    - **Workflow**: allowing proxy to approve workflows.
    - **Approve Proxy**: allowing proxy to approve any transactions affecting the proxy user.
    - **Change Data**: allowing proxy to change or update data.
    - **Change Variant**: allowing proxy to modify the Logic Variant that the original user normally accesses.
    - **Change Logic**: allowing proxy to modify any logic on behalf of the original user.
- **Timed Access**: is used to define periods of allowed or denied access for various access types.

The Functional Access Security process offers the ultimate level of control over access to all component objects of the meta-data EIS application, providing dynamic access control whenever the application is extended by the definition by vendors or users of new application logic or variant logic.

5.7 Advanced Operation Features

Many of the features of the meta-data EIS application are based on the key structural elements of the application logic as they correspond to the similar layers of standard application development, as has been portrayed in the previous sections.

The meta-data EIS application offers additional advanced capability beyond the replication of EIS application functionality including:

- **User Customisation via Variant Logic**: the capability for users to alter the pre-defined application logic to their own definition.
- **User Defined Integrated Application Workflow**: provides a more targeted focus on defining the steps that are required to achieve a required outcome and specifying the appropriate logic that will support those requirements.
- **Multi-Lingual Applications and Text Translation**: providing multi-lingual options for all application logic components, plus designing in a multi-lingual capability for the data in any meta-data EIS application.

These additional advanced features are described further in the following sections.

5.7.1 User Customisation via Variant Logic

In addition to business users defining and creating their own application logic, another key aspect of the meta-data EIS application is the capability to alter the pre-defined application logic as what I term Variant Logic, to become a variation of the application logic for a modified purpose.

Variant Logic can be applied to any object defined in a meta-data EIS application. It can be defined by any user and can be executed by any user as an alternative to the standard application logic. Of course, there are available security options that can be applied to manage all of these options, particularly where there is

important application logic that must be adhered to. The Variant Logic model is described in 5.4.5 Variant Logic .

5.7.2 User Defined Integrated Application Workflow

The meta-data EIS application includes an integrated application workflow capability that users can define themselves with any required logic, accessing any required application objects, without reliance on a third party workflow manager product. An additional benefit of the meta-data EIS application is that external workflow manager products can access the internal objects of the meta-data EIS application via standard web services for object invocation or access.

The meta-data EIS application provides extensive support for defining application workflows via;

- **Content**: existing screen components can be readily re-assembled and reused to create specific wizard workflow sequences without coding.
- **Access**: managing which users have appropriate access to data, transactions and authorisations is crucial to business operations.
- **Processing**: specify interaction sequences that apply to the transmission of information and any subsequent modification, updates, rejections etc.
- **Authorisation**: allowing the combinations of authorisations that can apply to the information, and the authorisation or rejection criteria.

The full model is described in 5.4.4 Application Workflow .

5.7.3 Distributed Execution Options

A perennial problem with larger decentralised organisations is the integration and transfer of data between business units, including the progressive processing and rollup of data between hierarchical business levels. When business units also utilise different EIS applications the data sharing can be more problematic due to the need for additional business logic verification.

The MDEIS framework can provide significant aids to resolving common inter-business unit data authorisation and transfer requirements. These aids can be utilised whether the different business units utilise the same or different meta-data EIS applications, although the most straightforward and simplest options can be more readily established when the meta-data EIS applications are identical.

Rather than requiring the traditional hard-coding of purpose built data retrieval, processing and transfer applications, potentially for each end of each data transfer link throughout the organisation, business units that execute the common runtime execution environment need only to identify the model objects that are to be shared, transferred or updated between the two applications and the runtime engine will then automatically manage the ongoing required transfers.

I define the Distributed Components (DC) of the MDEIS framework that could seamlessly provide advanced integration services such as: data replication, transfer and transformations; centralized authorization and distribution of core identity data; sharing and deployment of modified logic model elements; and workflow integration between application instances.

The distributed components can implement enterprise wide information sharing and access and minimize the need to develop specific data transfer, integration and processing features. They implement the required data exchange and processing functionality with minimal additional logic definition (note these are not customizations) and without the need for additional or specific data transfer and processing utilities.

The MDEIS application achieves this by the implementation of identical or similar core meta-data application logic at every distributed site across the enterprise - analogous to the installation of common application software for each site.

The distributed components can then be invoked at any site's MDEIS application instance by defining any of the following types of Distribution Execution Requests (DER) that will then operate between any groups of MDEIS application instances:

- **Data Replication**: (DR) defines the automated transfer of transaction or summary data between MDEIS instances.
- **Key Authorization**: (KA) defines a distributed schema for obtaining key, identifier or sequence based data from a pseudo master MDEIS instance simulating a distributed authorization hierarchy or other virtual topology of MDEIS instances.
- **Logic Variant**: (LV) defines the transferring of a locally defined Logic Variant to other MDEIS instances for local execution.

- **Workflow Trigger**: (WT) defines a pseudo master MDEIS instance to automatically escalate defined application workflow objects requiring transaction authorization beyond local authorization limits.

Figure 37 is a highly condensed overview of how the distributed components are modelled.

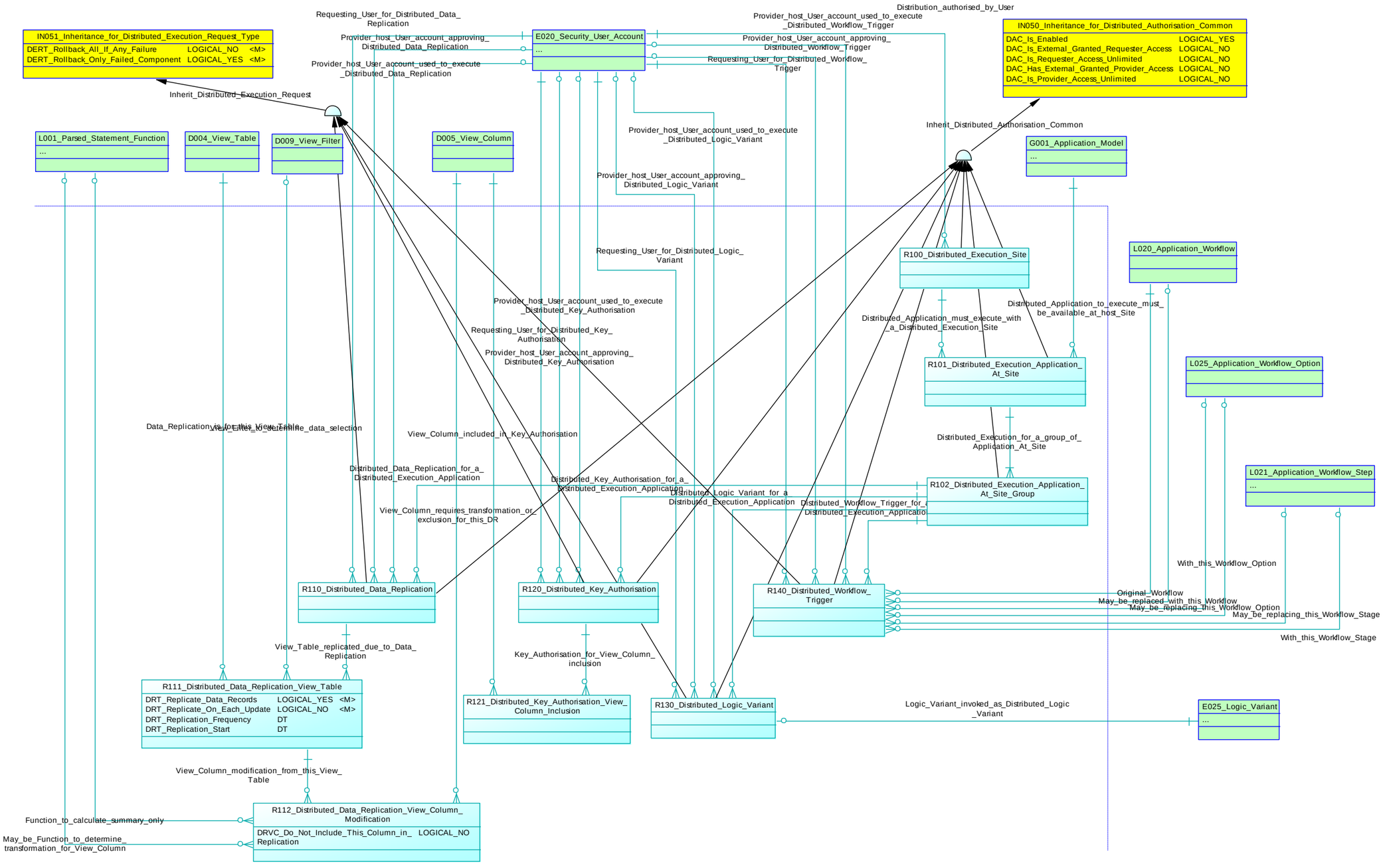


Figure 37 – Overview class diagram of the Distribution Execution Requests model objects

Any combination of these DERs with each requiring only minimal definition at each of the involved MDEIS instance nodes, denoted the Master and Slave nodes, will be executed automatically by the MDEIS runtime components of each MDEIS instance.

The overall authorization topology is composed of pairs of MDEIS instances called Master and Slave nodes. A Master node is the MDEIS instance that defines the requirement via the definition of a DER type and sends it to a Slave node for its local execution and possible transfer of information back to and between the Master node. Each Master and Slave node pair must first be granted mutual privileges to accept DERs from the other node.

The classification of whether an instance is a Master or Slave node is a term relative to each DER transaction authorization - any instance can be both a Master and Slave node, both mutually and to other instances.

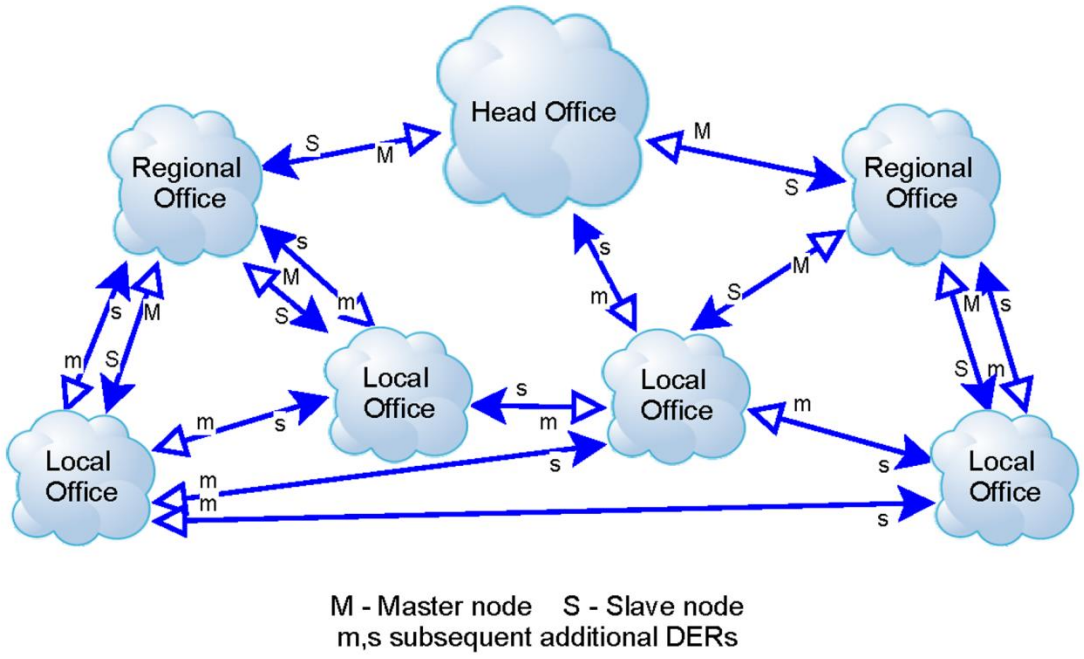


Figure 38 – Example multiple ad-hoc authorization nodes for a de-centralized organization

Figure 38 illustrates where a few additional DERs have subsequently been authorized between any required sites acting as either the Master or Slave node as

required. Ultimately such a DER mapping could be based on hundreds or thousands of individual DERs across a large enterprise.

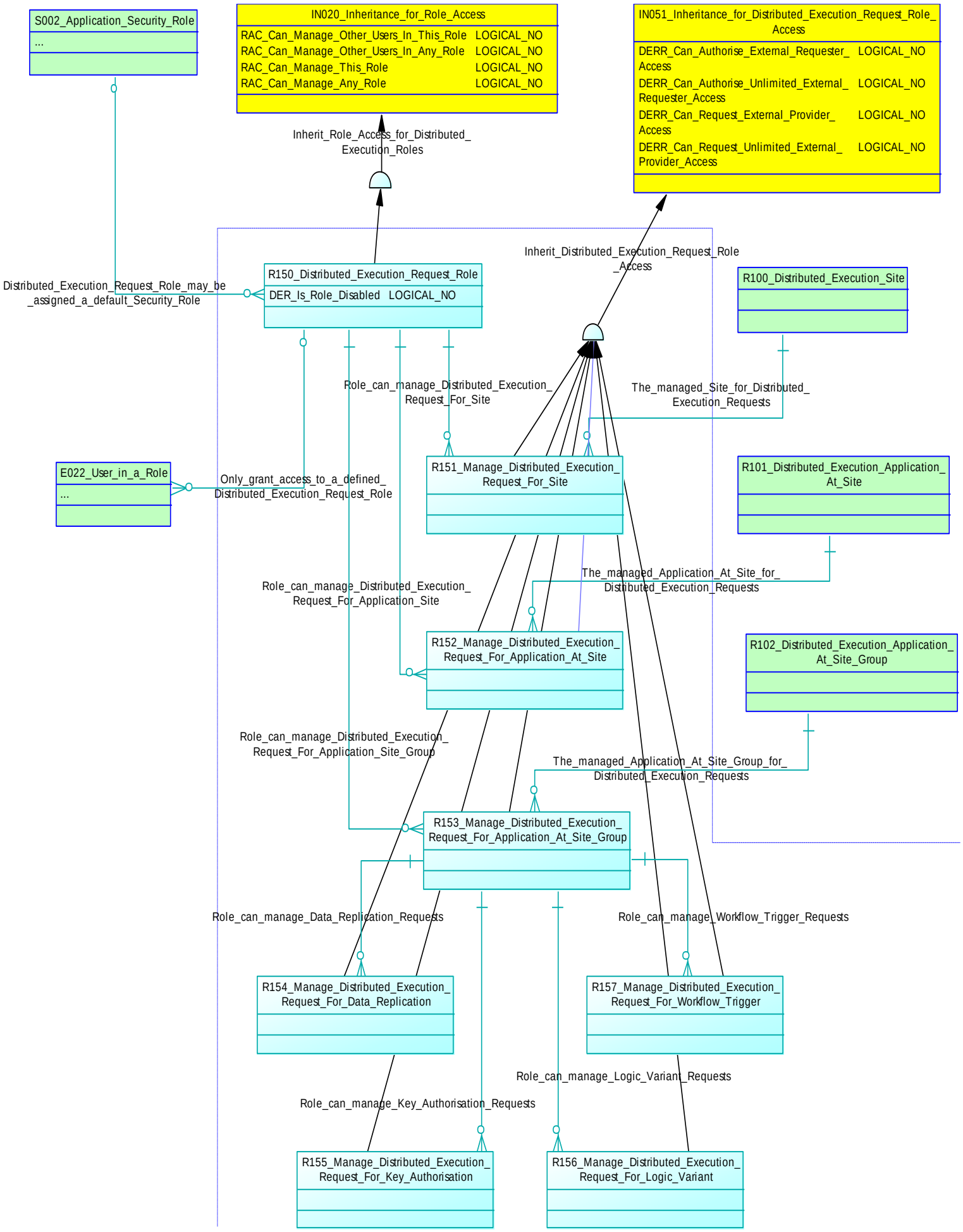


Figure 39 – Overview class diagram of distributed components authorization

Figure 39 provides an overview of the authorization classes for the distributed components of the MDEIS application model which allows for assigning authorizations to generate or approve individual DERs as well as in establishing and approving the inter-instance relationships.

5.7.3.1 Data Replication DER

The Data Replication DER is used to define an automated transfer of data from the Slave node back to the Master node. The required data for each DR can be either a copy of each transaction record or a calculated summary of data records based on a defined collation function. The incoming replicated data is treated by the incoming instance as authentic data - simply sourced via a special batch process rather than by individual user entry.

The DR operates primarily on a View Table basis, which is a collection of View Columns (analogous to aliased database tables). For each selected View Table the dimensions of the data replication must be determined from these options:

- Select whether data records are to be replicated or if only a single summary record is to be created and updated. If only a single summary record is selected then a Function must be defined for each included View Column to determine the value to be updated.
- The DR performs an update based on only the data that has changed since the last replication event which can be based on choosing to replicate after each host record update or on a defined frequency or period basis.
- Select if any modifications are required for any View Columns which can be; to not include a View Column in the replication, or to specify a transformation Function to apply to the View Column value.

The DR can be used to automatically effect distributed operations such as posting sales transactions from point of sale sites to a regional head office or posting transaction summaries from local offices to head office without the need to modify or customize the underlying MDEIS application logic or code.

One important constraint on the effective operation of the Data Replication DER is that a Master node will now be replicating data from another instance or site. In most cases this new data will need to be identifiable with an appropriate Application Site Identifier to correctly separate and identify the new remote site’s data from the current site’s data – this Application Site Identifier must be defined as part of the

application logic, either explicit to the data design (preferred) or defined as a View Column transformation (exercise caution).

Internally, all local data (and model meta-data) is identified with an Internal Site Identifier to differentiate data that is local and thus treated as authentic application data by the instance, compared to other supporting or related data that has been replicated from another MDEIS instance or site which therefore will be identified by setting the Internal Site Identifier to indicate its source as the external site.

In Figure 40 the DR View Table is the SALE_TRANSACTION which has the basic sales history from a site that is desired to be replicated at a Master node instance. If the SALE_LOCATION entity did not exist then the DR would simply replicate the sales transaction into the SALE_TRANSACTION View Table on the Master node instance potentially mixing and conflicting the sales transactions with those from the local Master node instance and possibly from other Slave node instances. To avoid this problem the application logic needs some form of Application Site Identifier to be applied to the original View Table which should be applied at every application instance – in this case the SALE_LOCATION entity via a mandatory foreign key – which could either be defined as part of the original core application logic or applied later as an example of a Logic Variant (which could itself be an example of a DER).

Note that Figure 40 also depicts an inherited Distributed Execution Site to all application entities – this is the Internal Site Identifier that is automatically applied to all data and meta-data within the data and model structures to support the co-existence of replicated distributed data whilst maintaining overall application data integrity. Hence the SALE_TRANSACTION data will be replicated on the Master node instance with the Internal Site Identifier set to the Master node instance and the Application Site Identifier copied as already set by the Slave node instance to the Slave node instance's Application Site Identifier. As the DR must also replicate data related to the target View Table (SALE_TRANSACTION) then the related data from SALE_LOCATION and PRODUCT are also replicated but these will have the Internal Site Identifier set to the originating Slave node instance to avoid conflict with either the Master node or other replicated Slave node data.

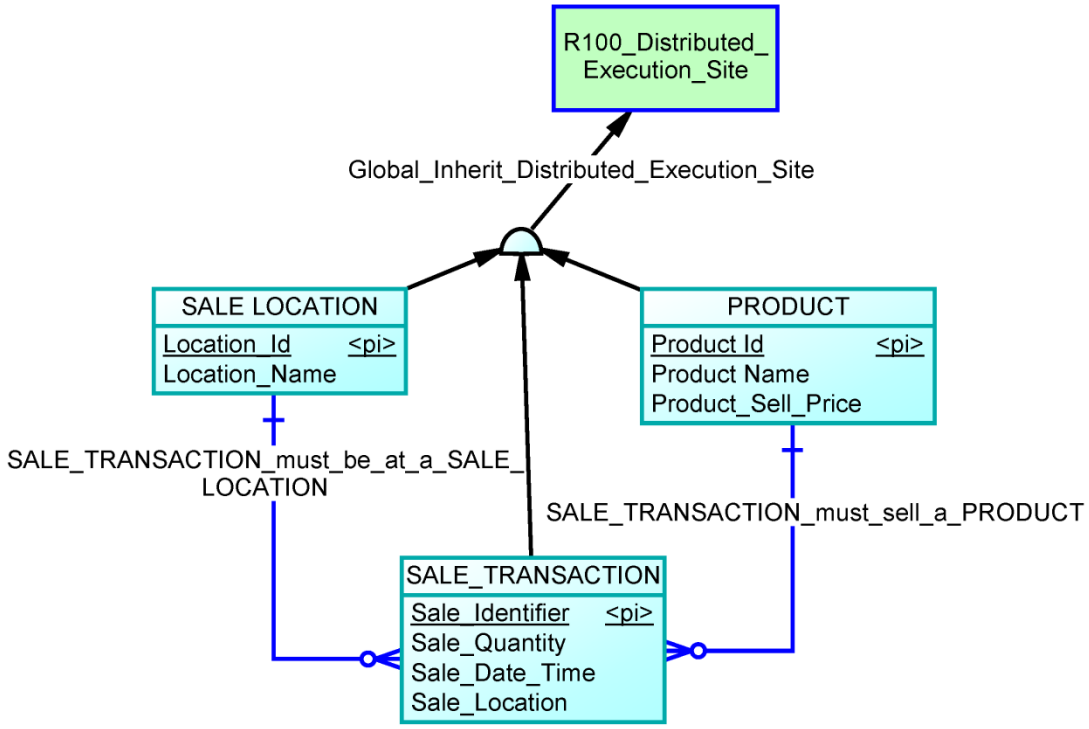


Figure 40 – Illustration of Internal Site Identifier and Application Site Identifier

As the incoming replicated View Table data is treated as genuine transactional data by the Master node instance (i.e. its Internal Site Identifier will be set to the local Master node site identifier while the Application Site Identifier will be some transformation or alias of the external Slave node site identifier) then all data related to the replicated View Tables must also be replicated as needed to ensure data integrity is maintained on the Master node instance.

However, this related data is not treated in exactly the same way – the related data is recorded in the associated Master node View Tables (which already exist due to the similar or identical application models involved in the DER) but with the Internal Site Identifier set to the external Slave node site identifier – all internal schema relationships also inherit the Internal Site Identifier attribute as a hidden dependency to support replicated site data. These additional transactions to replicate the related data from the external Slave node site onto the local Master node site are performed automatically and directly as all of the related data has already been authenticated on the external Slave node site instance.

5.7.3.2 Key Authorization DER

The Key Authorization DER defines a distributed schema for obtaining key, identifier or sequence based data from a pseudo master MDEIS instance simulating a distributed authorization hierarchy or other virtual topology. The KA provides the ability for a Master node to become the distributed source of data for some key data columns to selected Slave nodes. This source of this key data can be defined on the Master node on the basis of pre-prepared lists of data or generation rules to be used for each Slave node. When the Slave node's MDEIS application requires a new value for the nominated key data, instead of generating or entering the data locally on the Slave node a distributed request back to the Master node is performed to retrieve or generate the required data.

The KA operates on a View Column basis and would typically be defined only for important identifiers, unique values, or high security data that the organization requires to be served from a centralized (or at least remote) MDEIS source instance to other MDEIS instances. The KA data from the Master node is provided to any requesting Slave node which treats the data as though it were entered or defined directly at or by the requesting Slave node.

The Master node can either serve existing predefined data or can utilize standard MDEIS auto-generation Functions to provide the rules governing the creation of any new data for the defined View Columns. This utilizes a standard feature of the MDEIS application framework whereby any View Column can be allocated these generation rules (see Figure 41).

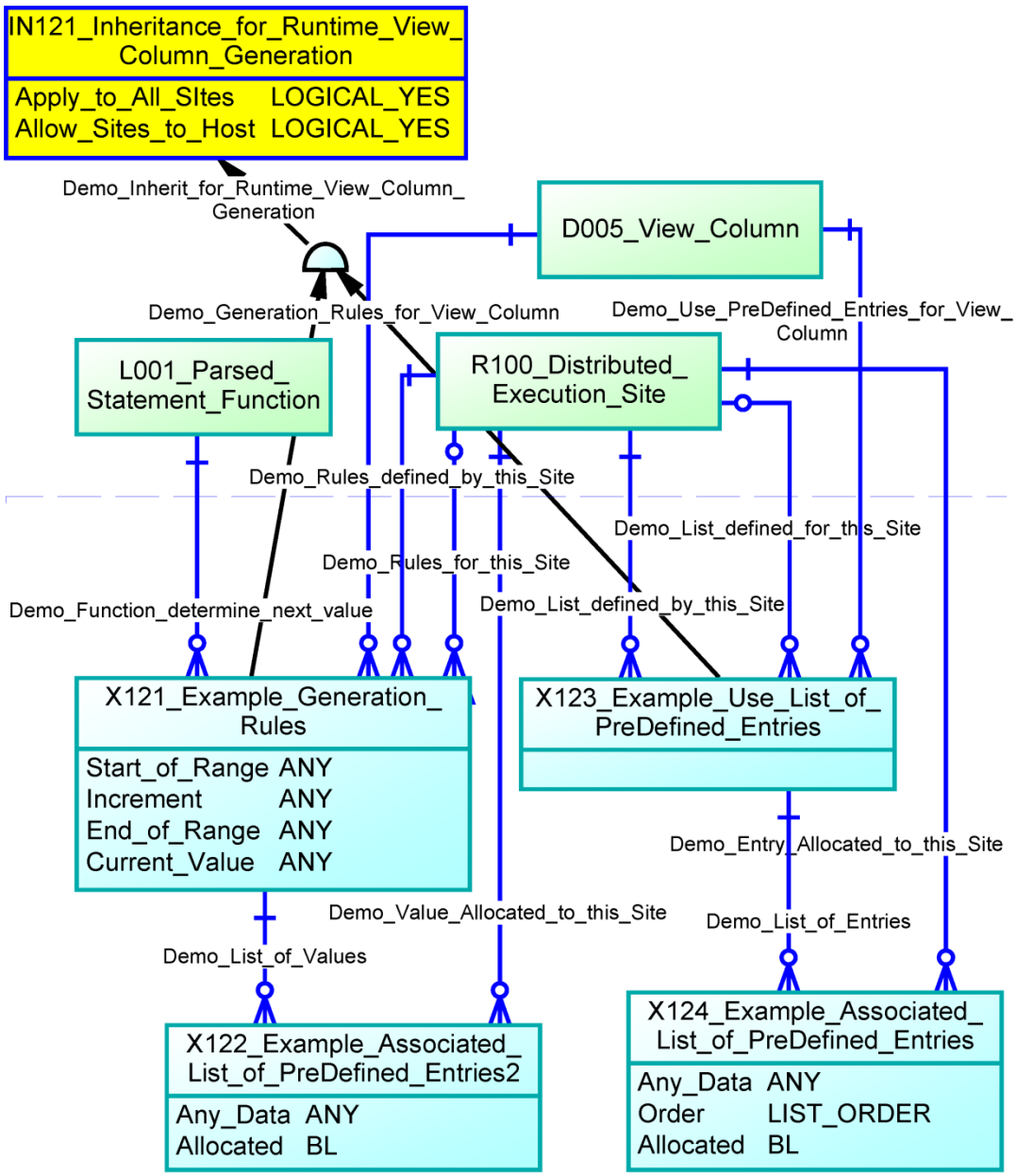


Figure 41 – Overview of generic View Column Generation options

When the KA is authorized the option exists for the rules to be distributed to the Slave node's MDEIS instance for both performance efficiency and optional network failure continuance. This option is only available if the rules are site specific – global (or shared) rules always require runtime distributed authorization from the Master node.

The KA can be used to establish hierarchical or centralized definition, management, distribution and allocation of pre-defined identifier data such as; employee identifiers, serial numbers, sequence or transaction identifiers, or other high

security data without the need to modify or customize the underlying MDEIS application logic or code in any way, other than defining the KA DER.

5.7.3.3 Logic Variant DER

The Logic Variant DER defines the transferring of a locally defined Logic Variant to other MDEIS instances for local execution. A Logic Variant is the unique capability for third parties and end users to define and create their own application logic to supplement or replace a vendor's pre-defined MDEIS application logic to become an alternate variation of the application logic [190].

As the MDEIS application instances are identical or similar then a Logic Variant from the Master node will be expected to execute on other Slave node MDEIS application instances to provide the identical functionality. The MDEIS application framework provides the capability to define Logic Variants (as customizations) throughout the application model, the LV DER provides the ability to share and deploy these customizations with other instances executing the same MDEIS application.

The LV operates on a defined Logic Variant that would be expected to have already been utilized and verified in the current Master node instance. When authorized to be deployed to Slave nodes the Logic Variant definition will be copied to each Slave node for either global usage or on a Roles basis. Once installed locally at the Slave nodes the Logic Variant can then also be assigned to other local users as required although it cannot be modified locally on the Slave node. Any subsequent modifications to the original Logic Variant at the Master node will be broadcast to each Slave node defined in the DER.

The LV can be used to share and distribute any Logic Variant customization between MDEIS application instances where the new application logic segment serves a useful or mandatory purpose (in the case where Logic Variants that are applied globally to all users). A Logic Variant can provide any defined functionality.

5.7.3.4 Workflow Trigger DER

The Workflow Trigger DER defines a pseudo master MDEIS instance to automatically escalate application workflow objects requiring transaction authorization beyond local authorization limits. The appropriate element of the workflow will then be transferred to and executed on the Master node returning the

result back to the Slave node for further processing and execution of any remaining workflow elements.

The WT has three options to define the escalated alternate execution on the Master node for the defined Application Workflow elements:

- Application Workflow: Completely replaces execution of the Application Workflow onto the Master node.
- Application Workflow Option: An Application Workflow is composed of one or multiple Application Workflow Options – only one option will ever be executed based on a conditional test. This WT option transfers the execution of one Application Workflow Option to the Master node.
- Application Workflow Step: An Application Workflow Option is composed of one or multiple Application Workflow Steps – the set of Steps representing any serial or multi-path parallel sequence. This WT option transfers the execution of one Application Workflow Step to the Master node.

A key consideration is how the Master node knows what object from the Slave node is the triggering subject of the original workflow, plus how the Master node provides the result and decision tracking information back to the Slave node. Note that the Application Workflow already exists on both the Master and Slave nodes as they are executing the same MDEIS application, as do all of the application data structures. Recall also the mechanism that the Data Replication DER utilizes to replicate all related data between the Slave and Master nodes to provide for full supporting transaction data and maintain data integrity. The identical replication process utilizing the Internal Site Identifier is used for the WT DER.

Firstly, the triggering object on the Slave node and related data are replicated to the Master node, as well as any other data objects that are referenced by the relevant workflow element object. The workflow element will then be executed on the Master node and the result recorded.

Finally, the result of the workflow element plus all associated workflow tracking and audit data (including the authorized users and their decisions) from the Master node must be replicated back to the Slave node. The result from the Master node is then integrated into any remaining workflow elements on the Slave node and the workflow execution continues to its ultimate conclusion on the Slave node. Figure 42

illustrates the execution pathways and replications that occur on the Slave and Master nodes during the execution of a Workflow Trigger DER that replaces an Application Workflow Step.

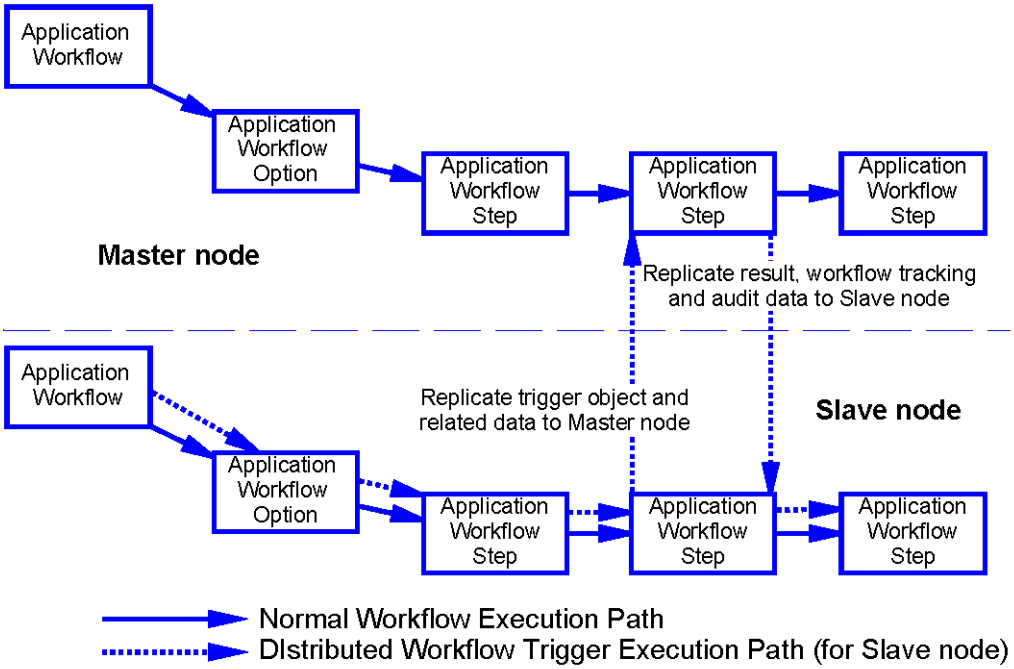


Figure 42 – Workflow Trigger DER executing an Application Workflow Step

To clarify, tracking and results data that are created due to the normal execution of the workflow element on both the Slave and Master nodes is treated normally as genuine data on the local MDEIS instance i.e. the Internal Site Identifier is set to the executing instance. Replicated data from another MDEIS instance that supports the execution of the workflow has the Internal Site Identifier set to the originating instance to maintain local data integrity.

The WT can be used to establish a selective hierarchical authorization of Application Workflows such as; high value financial transactions that require a higher regional or head office authorization, or key policy document amendments or any form of important information to be routed through higher corporate review workflows.

5.7.3.5 Verifying DER Compatibility

There are two pre-requisites for creating DERs; that the same core Application Model is executing at both the Master and Slave node MDEIS instances, and that the

Application Model needs to be identical or similar (enough). Clearly an identical Application Model would have no differences at all in any model elements – it would be analogous to an identical versioned set of executable applications and configurations in the traditional sense – and DERs would readily deploy and execute between these identical instances.

Our compatibility algorithm detects any clear syntactic mismatches between any of the required model objects which will abort the DER due to incompatibility. However, it is not always so clear to be able to always guarantee semantic compatibility - it is possible, due to valid module or partial application model updates, that there can be some model objects in the Slave node that are of a later subjective version than the Master node and some that are of an earlier subjective version, and vice versa simultaneously. The verification process will also advise if either or both the Slave and Master nodes are recommended to be upgraded for a greater likelihood of semantic compatibility.

The Distributed Execution Request capability of the MDEIS application framework provides some unique benefits to organizations that choose to operate similar applications throughout the enterprise in a distributed execution topology.

The DER capability is a recent capability extension to the model and framework to address integration issues of distributed applications as an economic and opportunistic core feature set addition. As implemented in a production instance each of these DERs could be implemented directly between sites' MDEIS application instances through simple definition of the runtime DER model logic, requiring no additional combinations of third party utilities, middleware or custom programming, and are managed within a single definition and authorization context, the MDEIS application instances.

5.7.4 Multi-Lingual Applications and Text Translation

Historically, the majority of applications have not been developed with the ability to provide multi-lingual options, in most cases separate versions have been required to be maintained with the alternate language text coded in every instance. Advances have been made with developer toolsets that assist with a level of separation of object text from the code so that separate language files can be maintained, offering greater flexibility.

The meta-data EIS application provides the full capability for multi-lingual presentation of both the application logic and the application data as:

- **Application Text**: all text components of the meta-data objects are separated from the meta-data objects to provide language specific alternatives.
- **Text Formats**: the design of the display attributes allows for defining the orientation, sizing and entry of the alternate language sets.
- **Alternate Data Translation**: allowing for multi-language translation copies of textual data.

The design aspects of the multi-lingual options for the meta-data EIS application are described in the following sections.

5.7.4.1 Fundamental Multi-Lingual Entity Schema

There is a key fundamental aspect of the multi-lingual capability that applies to all meta-data objects of the meta-data EIS application providing for the key object descriptions, user help and user manual notes for alternate language sets.

The design has been included as part of 5.2.1 Generic Distributed Temporal Meta-Data Inheritance which includes other generic aspects of the model.

5.7.4.2 Generic Visual Element Meta-Data Entity Schema

The visual elements of the meta-data EIS application also have language dependent features that affect the display of each visual element. Primarily based around the sizing of the visual element as can change depending on the overall sizing differences between alternate language sets, orientations, fonts and nominal sizes.

Figure 43 depicts a sample of the additional relationships that need to apply to the visual element meta-data entities in the meta-data EIS application model in order to provide for alternate multi-lingual language definitions. Whilst these relationships are defined within the general model design, they are not usually depicted in the functional design excerpts that are illustrated in most areas of this thesis.

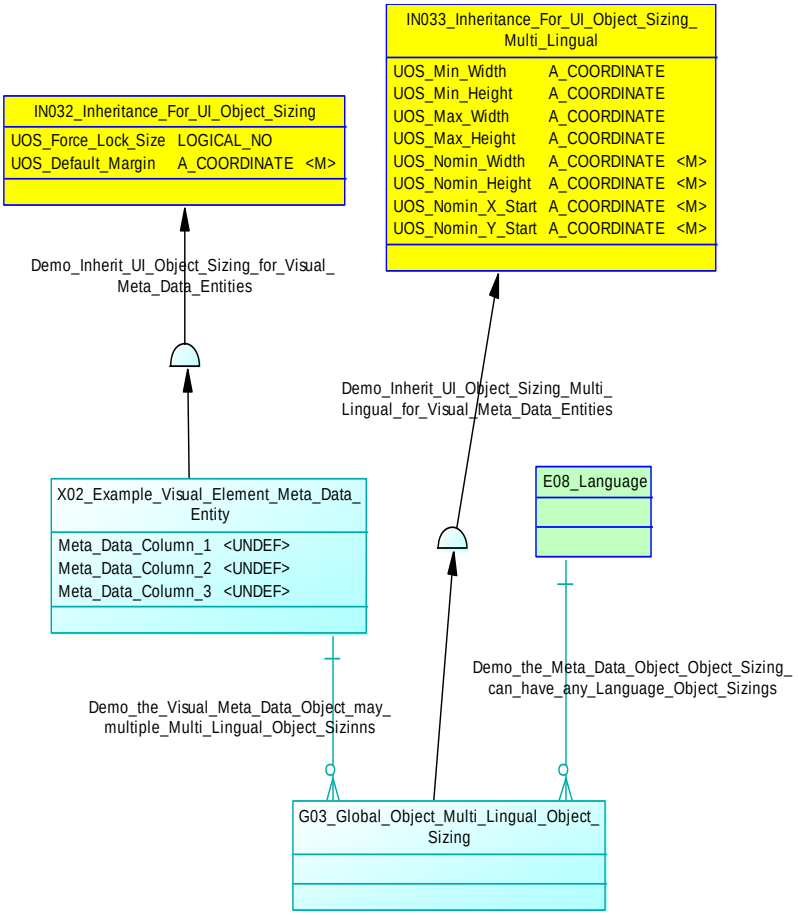


Figure 43 – Visual Meta-Data Entity Schema

The Generic Visual Element Meta-Data Entity Schema design uses the following entities to model the required functionality:

- **Inheritance For UI Object Sizing**: is inherited to all visual meta-data entities. It provides the language independent sizing options.
- **Example Visual Element Meta Data Entity**: is a sample meta-data entity to represent the relationships that the visual element meta-data entities will have as background relationships. E.g. Canvas, Freeform Panel etc.
- **Inheritance For UI Object Sizing Multi Lingual**: is inherited to the Global Object Multi Lingual Object Sizing meta-data entity. It provides the specific meta-data object sizing options.
- **Global Object Multi Lingual Object Sizing**: maintains the above inherited object sizing data for each visual element meta-data object on a

per language basis to provide a multi-lingual solution for any meta-data EIS application.

- **Language**: is a list of languages used for localisation options. This is modelled as a relationship from the Global Object Multi Lingual Object Sizing meta-data entity to identify the specific language for a meta-data object's multi-lingual definition.

The Generic Visual Element Meta-Data Entity Schema design is one of several additional multi-lingual requirements, focussing on the visual elements.

5.7.4.3 Generic UI Object Text Display Meta-Data Entity Schema

Many of the UI Object visual elements of the meta-data EIS application also have language dependent features that affect the display of textual data for that UI Object. Primarily based around the text orientation, font, and hover text and audio aspects of alternate language sets.

Figure 44 depicts a sample of the additional relationships that need to apply to these UI Object meta-data entities in the meta-data EIS application model in order to provide for the display of alternate multi-lingual language definitions. Whilst these relationships are defined within the general model design, they are not usually depicted in the functional design excerpts that are illustrated in most areas of this thesis.

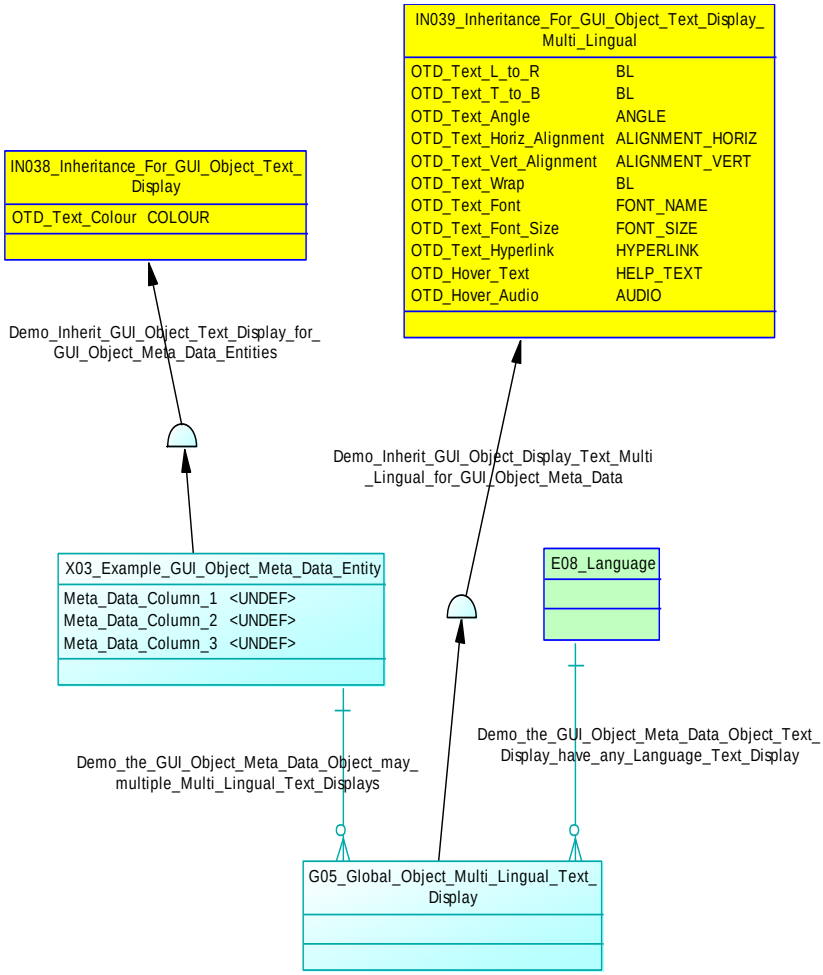


Figure 44 – UI Object Text Display Meta-Data Entity Schema

The Generic UI Object Text Display Meta-Data Entity Schema design uses the following entities to model the required functionality:

- **Inheritance For UI Object Text Display**: is inherited to UI Object meta-data entities that display data. It provides the language independent colour options.
- **Example UI Object Meta Data Entity**: is a sample meta-data entity to represent the relationships that these UI Object meta-data entities will have as background relationships. E.g. Navigation Panel, GUI Text etc.
- **Inheritance For UI Object Text Display Multi Lingual**: is inherited to the Global Object Multi Lingual Text Display meta-data entity. It provides the specific meta-data object text orientation, font, hyperlink and hover text and audio aspects options.

- **Global Object Multi Lingual Text Display**: maintains the above inherited object formats data for each UI Object meta-data object on a per language basis to provide a multi-lingual solution for any meta-data EIS application.
- **Language**: is a list of languages used for localisation options. This is modelled as a relationship from the Global Object Multi Lingual Text Display meta-data entity to identify the specific language for a meta-data object's multi-lingual definition.

The Generic UI Object Text Display Meta-Data Entity Schema design is one of several additional multi-lingual requirements, focussing on the UI Objects.

5.7.4.4 Generic UI Object UI Entry Meta-Data Entity Schema

A few of the UI Object visual elements of the meta-data EIS application also have language dependent features that affect the entry of textual data for that UI Object. Primarily based around the entry mask and default text aspects of alternate language sets.

Figure 45 depicts a sample of the additional relationships that need to apply to these UI Object meta-data entities in the meta-data EIS application model in order to provide for the entry of alternate multi-lingual language definitions. Whilst these relationships are defined within the general model design, they are not usually depicted in the functional design excerpts that are illustrated in most areas of this thesis.

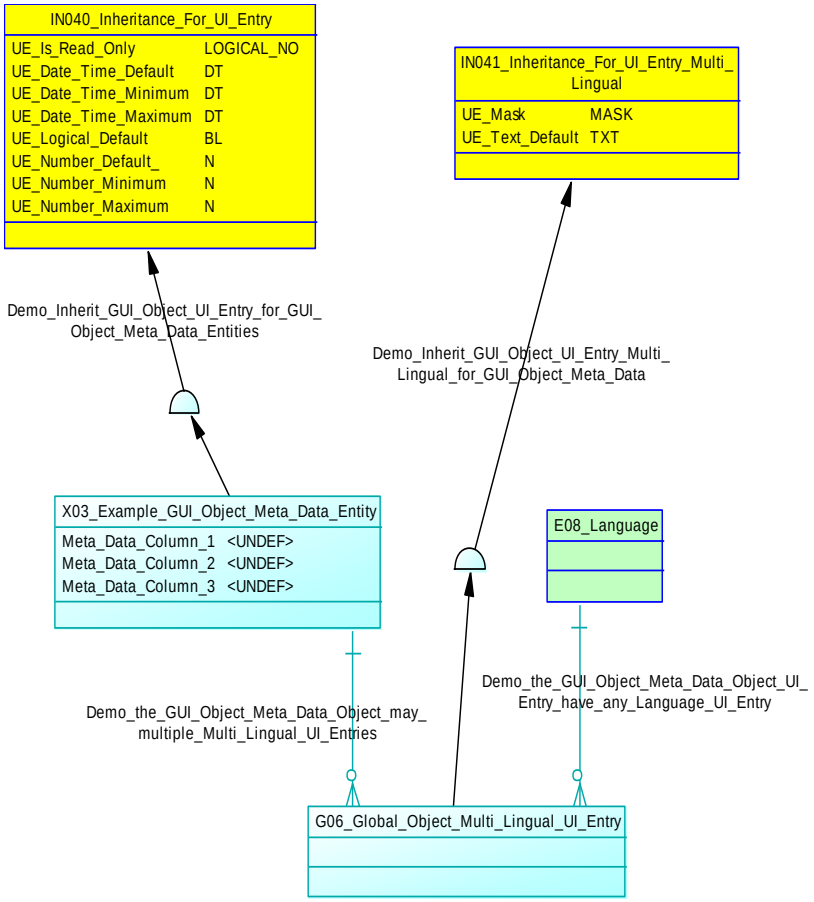


Figure 45 – UI Object UI Entry Meta-Data Entity Schema

The Generic UI Object UI Entry Meta-Data Entity Schema design uses the following entities to model the required functionality:

- **Inheritance For UI Object UI Entry**: is inherited to UI Object meta-data entities that display data. It provides the entry status and non-language dependent defaults and ranges options.
- **Example UI Object Meta Data Entity**: is a sample meta-data entity to represent the relationships that these UI Object meta-data entities will have as background relationships. This only applies to the GUI Text Box and GUI Selection entities.
- **Inheritance For UI Object UI Entry Multi Lingual**: is inherited to the Global Object Multi Lingual UI Entry meta-data entity. It provides the specific meta-data object entry mask and default text aspects options.
- **Global Object Multi Lingual UI Entry**: maintains the above inherited object formats data for each UI Object meta-data object on a per language

basis to provide a multi-lingual solution for any meta-data EIS application.

- **Language**: is a list of languages used for localisation options. This is modelled as a relationship from the Global Object Multi Lingual UI Entry meta-data entity to identify the specific language for a meta-data object's multi-lingual definition.

The Generic UI Object UI Entry Meta-Data Entity Schema design is another of several additional multi-lingual requirements, focussing on the UI Objects.

5.7.4.5 Multi-Lingual Entity Schema for Data

The option is also provided for the application data managed by the meta-data EIS application to be stored as multi-lingual translation options supporting operational environments where an organisation's global users of the meta-data EIS application are accessing the same global data.

The design has been included as part of 5.2.2 Generic Distributed Temporal Data Inheritance which also includes other generic aspects of the model.

5.8 Conclusion

The detailed models described in this chapter address each of the key requirements raised in Chapter 4 - Conceptual Framework for Temporal Meta-Model for Enterprise Information Systems, demonstrating how the meta-data EIS application can provide the fundamental advances in providing the closer integration between EIS applications and the business environment.

The objective of the creating similar EIS applications and functionality with meta-data EIS applications is a key requirement and by itself has been shown to provide substantial overall lifecycle benefits. However, as has been shown. there are significant other benefits to be realised in the areas of user empowerment that have not really contributed to our estimates of the meta-data EIS application lifecycle optimisation assessments, yet by themselves can offer enormous potential benefits.

The ability for users to define their own application workflows for both process flow and authorisation provides the opportunity for personal workplace optimisation as well as the security of adequate authorisation.

Variant Logic is perhaps one of the highest potential optimisers, allowing users to customise any aspect of the defined meta-data EIS application. This empowerment option to users to develop or modify application logic to best fit their own processes yet maintaining compatibility within the organisational environment can offer almost limitless versatility.

As globalisation continues, the ever growing need for overcoming the diversity of international user groups and divisions of organisations by using the same EIS applications can become a reality via the inherent multi-lingual meta-data and data options of the meta-data EIS application, rather than the use of disparate and disjointed separate solutions subject to the concerns of timely and accurate data and workflow integration.

In this chapter, I have described the design model for the meta-data EIS application that can achieve all of the stated requirements. The following chapters will utilise these models to describe the practical implementations of the meta-data EIS application.

Chapter 6 - Agile Platform for Dynamic Systems Change Management

6.1 Introduction

A domain specific model for EIS applications, whether in the traditional form as software source code or defined as the higher level meta-data EIS application model as described in this thesis, requires a separate execution environment that transforms the model into operational use.

In the traditional application development environment compilers are used to verify the syntax of the source code and produce an executable machine language file or a transitionally coded model file that will invoke the required functions of the full runtime environment as required during execution – in reality most modern compiled applications will execute similarly due to the progressive proliferation of modern execution frameworks such as Java, .Net and additional third party software providers which can provide the majority of processing features as pre-compiled libraries – the EIS application model similarly requires the support of its temporal runtime framework for execution.

The runtime engine for the meta-data EIS application model verifies the integrity of the defined model and provides the matching executable functionality for all

modelled elements. This thesis does not prescribe any specific development architecture of the runtime engine although does suggest candidate options, as any deployment option can be provided for the EIS applications. The general requirement for any runtime engine is that full compatibility with and support for all features of the meta-data EIS application model is maintained, ensuring that the same model can be executed by any individually architected runtime engine and process the inputs to obtain identical outputs.

A key desirable aspect of the meta-data EIS application runtime engine is that it is able to dynamically respond to model changes i.e. the current meta-data EIS application model must be the source for the runtime engine and not require the often lengthy and convoluted compilation processes of traditional application development, nor their typically manually and delayed deployment of executables, particularly when customisations have been made for the end user. This dynamic model response provides the substantial benefits expected from such features as Variant Logic which allows users to add to and redefine the core application logic for their own local conditions.

This chapter describes the runtime specific aspects required to support and execute the meta-data EIS application model in terms of overall functionality and any additional model segments unique to the runtime environment. Where the model definition concentrates on defining what the application is intended to logically do, the runtime engine must provide a solution for how this will occur.

The sections in this chapter do not delve to the level of a 1 to 1 description of the detailed functionality required for each model element, as these finer details are provided in both the model definition (see Chapter 5 - Instant Interaction EIS System Modeller), the function definitions (see Chapter 8 - Universal Access to Temporal Meta-Data Framework for EIS in the Cloud) and encapsulated in the more detailed designs referred in the thesis appendix as attachments.

The additional design detail for the runtime engine in this chapter focus on the additional major architectural considerations that must be supported in addition to each atomic element's specific functionality. These include:

- **Temporal Data and Meta-Data Management**: features that support a varying historical or temporal basis for the meta-data EIS for both data and application logic.

- **Automated Version Control and Deployment**: features that facilitate rapid, even immediate, deployment of the changing meta-data EIS application into operational production use.
- **Transaction Management**: Transformations from the atomic transaction statement components into a format more applicable to the interface of the database system or transaction processing engines.
- **Execution and Logic Definer Security**: Ensure that only appropriately authorised actions, data transactions and logic changes are enacted.

In the following sections of this chapter, these architectural requirements are further expanded to detail their required runtime operations and interactions with the meta-data EIS application model.

6.2 Fundamental Runtime Features

As the meta-data EIS application model is a model for the requirements of typical EIS applications then the more fundamental and common features of such software applications must form the basis of the provided features of the runtime engine, to provide the user interface, logical processing and data transactions across the supported architectures.

Security access control is the common glue that binds the application elements and presents the components for authorised usage. Whilst there are unique features of the meta-data EIS application model that need to be supported, from the user perspective there should be little or no obvious difference to whether the EIS application being executed is based on traditional development or from a meta-data EIS application model. Various architecture options are provided to cater for alternate platform access.

All core model elements of the meta-data EIS application model must be directly supported by the runtime engine as described in Chapter 5 - Instant Interaction EIS System Modeller. Further model definitions are provided as the function definitions (see Chapter 8 - Universal Access to Temporal Meta-Data Framework for EIS in the Cloud) with the detailed model designs referred in the thesis appendix as attachments which provide the full specification of the design requirements. These additional design elements for the runtime engine in this chapter focus on the additional major

architectural considerations that must be supported in addition to each atomic element's specific functionality.

6.2.1 Runtime System Architecture Fundamentals

While the meta-data EIS application model itself is multi-layered and heavily focussed on abstraction in its design, this does not impose any preference or limitation on the flexibility or abstraction of a runtime engine or its components.

As the world rides the ongoing technology wave centred around the relative immediacy and progressive pervasiveness of the internet and its ease of information accessibility, there can still be a multitude of different architectures that can be utilised for the meta-data EIS application runtime engine.

Many technology commentators would advocate application delivery via browser based or thin client technologies in order to maximise the availability of end user access, such as espoused by the proliferation and hopes of industry giant Google and its Chrome and Android operating systems and Google Apps. There are also key considerations such as the pervasiveness of suitable internet connections, and the speed, performance and overall user experience factor of the user interface that may suggest solutions composed of at least more localised user interface processing and data availability. Recent technology paradigm shifts exemplified by devices such as Apple's iPhone and iPad and the subsequent wave of new consumer tablets have highlighted the potential success of well-crafted specialist applications for a mass audience.

A general architecture that would suit the meta-data EIS application runtime engine is depicted in Figure 46 – Generalised Runtime Engine .

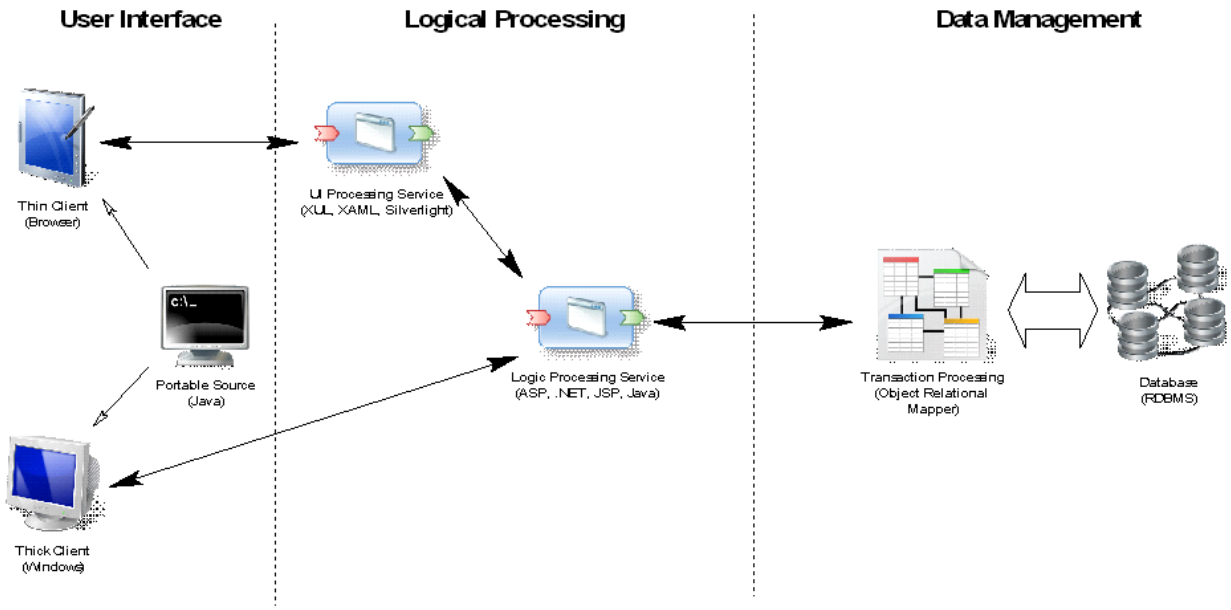


Figure 46 – Generalised Runtime Engine Architecture

In the above generalised architecture, it does not matter whether the user interface component of the runtime engine is based on a platform specific codebase, or multiple platform alternatives, or whether the user interface is delivered via a browser or similar technology. Similarly the logic processing can be performed more locally or from any remote service, and finally the transaction processing component can utilise local or remote processing and database engines – although it is expected that the long commonplace approach of separating the transaction processing and utilising standalone database engines would be a basic standard to be applied. Additionally, the use of object relational mapper technology to act as an efficient middleware layer in the transaction processing and minimise overall development time for the runtime engine would also be a reasonable expectation.

6.2.1.1 Fundamental Runtime Design Requirements

The key model components referred to in the appendix and attachments that relate to the generalised design of the runtime engine objects are:

- **Generic Temporal Meta Data Entity Schema**: general template to apply to all model objects that utilise temporal meta-data management.
- **Generic Temporal Data Entity Schema**: general template to apply to all external data storage objects that utilise temporal data management.

- **All Entities Basic Inheritance**: list all model objects and provides the general inheritance relationships, excluding specific functional inheritances that are listed further in this design.
- **PKFK Inheritance for Entities**: lists specific inheritances that are required to be manually specified due to a limitation in the design software used where the relationship object in foreign key relationships cannot be correctly inherited into the design.

The remainder of the internal design aspects that are not captured and provided as visually modelled design elements are listed in the attached full meta-data EIS application model design extraction document which also details the associated model business rules, parameters and options.

The following sections expand on further architectural options.

6.2.1.2 Fundamental Runtime Processing Algorithm

Based on the provided high level meta-data structures of the application model elements there are fundamental logical processes to be directly executed that effectively initialise the meta-data application session. Note that the definition of any specific user interface elements for a meta-data application is not mandatory i.e. there can be a non-visual meta-data application just as there can be a non-visual standard application.

The high level application processing is the first processing to occur for the meta-data runtime execution environment. The core algorithm is as follows:

- **Environment Initialisation**: status check to determine if the meta-data runtime engine is operational,
    - status check to determine if the meta-data runtime engine is operational,
    - verify any session initialisation parameters such as a requested Application Model, or identified User etc, to determine if the requested session can be enabled. The absence of any required session identification credentials will notify and request the entry or provision of appropriate credentials. Any request involving the use of common user interface objects, such as a user login prompt, is provided by runtime engine defined visual objects using the standard user interface rendering elements.

    - Upon credential verification a new session is enabled, or previous session restored (processing continues where it had been previously suspended).
- **Application Model Initialisation**: any defined Initialisation event is executed (optional). A non-visual meta-data application would only have non-visual Application Model event functions defined. Event functions can also invoke other visual and non-visual elements and functions.
- **User Interface Processing**: any defined visual structures, including any associated event functions, are progressively executed based on the defined user interface elements (see 6.2.2.2 User Interface Processing Algorithm). Ongoing application execution continues within this processing area based on the user session interactions, whether provided as localised user interactions or by remote command sequences, until the session is terminated by a request to temporarily suspend the session, or to fully terminate the session (see next processing step).
- **Application Model Completion**: any defined Completion event is executed (optional) before full session termination.

The additional logical processing and data processing functionality are progressively invoked as required by the non-visual and visual processing that is invoked within the above execution steps, based on the defined meta-data model structure.

6.2.2 User Interface Elements

The user interface layer is one of the most critical elements for general user acceptance as it represents the primary interaction that most users will have with the application. While the underlying business logic of the application must still be valid and efficient, a poor user interface presentation will deter users from adequate or efficient interaction.

The most common options for user interface presentation are the choice between; **thin client**, usually presented remotely and independently from the client workstation and often via a web browser providing the advantage of higher portability, or **thick client**, where the application is compiled for local execution on the client workstation and thus can usually provide a richer user interface experience.

In recent years there has been a significant escalation in the use of portable devices such as smartphones, tablets and pads with their associated application access although these tend to be a subclass of the thick client as their application or “apps” are developed specifically for execution on the device model or class such as iOS [191], Android [192]and Windows [193].

6.2.2.1 User Interface Design Requirements

The core user interface design requirements of the EIS application model exist independently of whether the runtime engine is implemented as a thin or thick client. The key implementation aspect is that the required user interface functionality is fully provided as per the design specification listed in the appendix Temporal Meta-Data EIS Application Model relating to the visual presentation design of the runtime engine.

The key relevant model components listed in the appendix attachments are:

- **Generic Visual Meta Data Entity Schema**: general template to apply to all model objects that constitute the visual presentation layer.
- **Generic UI Object Text Display Meta Data Entity Schema**: general template to apply to all UI Object visual model objects which display text, to facilitate multilingual representation and text storage.
- **Generic UI Object UI Entry Meta Data Entity Schema**: general template to apply to all UI Object visual model objects which prompt for the entry of text, to facilitate multilingual representation and text storage.
- **Visual Structure Visual Entity Structure**: the key overall design of all visual model objects and their inter-relationships.
- **Visual Structure UI Objects**: the expanded overall design of all visual UI Object model objects and their inter-relationships.
- **Visual Structure Advanced UI Objects**: the expanded design of the advanced visual UI Object model objects and their inter-relationships. These objects tend to have greater embedded or compound functionality over other simpler or atomic function UI Objects.

The remainder of the internal design aspects that are not captured and provided as visually modelled design elements are listed in the attached full meta-data EIS application model design extraction document which also details the associated model business rules, parameters and options.

6.2.2.2 User Interface Processing Algorithm

Based on the provided meta-data structures of the user interface elements there are fundamental logical processes to be directly executed solely on the nature of the visual structure, if there is any visual structure defined.

Processing of the user interface elements occurs after the initial application processing. The core algorithm for the user interface elements is as follows:

- **If First Canvas, then for each in order**: create and render the Canvas object based on the following:
    - **Canvas Initialisation**: any defined Initialisation event is executed (optional).
    - **If First Navigation Panel Used On Canvas, then for each in order**: create and render the Navigation Panel object based on the following:
        - **Navigation Panel Initialisation**: any defined Initialisation event is executed (optional).
        - **If First Navigation Panel Item, then for each in order**: create and render the Navigation Panel Item object based on the following:
            - **Navigation Panel Item Initialisation**: any defined Initialisation event is executed (optional).
            - **Render Navigation Panel Item**: if required as a currently displayed object.
        - **Render Navigation Panel**: finalise overall rendering including adherence to any object Alignment Rules.
    - **If First Freeform Panel Used On Canvas, then for each in order**: create and render the Freeform Panel object based on the following:
        - **Freeform Panel Initialisation**: any defined Initialisation event is executed (optional).
        - **If First UI Object Used On Freeform Panel, then for each in order**: create and render the UI Object object based on the following (note that each UI Object type will have its own type specific instantiation definitions:

- **UI Object Initialisation**: any defined Initialisation event is executed (optional).
- **Render UI Object**: including any type specific requirements and defined relationships with other visual objects, finalise overall rendering including adherence to any object Alignment Rules.

▪ **Render Freeform Panel**: finalise overall rendering including adherence to any object Alignment Rules.

o **Render Canvas**: finalise overall rendering including adherence to any object Alignment Rules.

- **User Interaction Processing**: following the initialisation event execution, definition and rendering of the required visual objects, subsequent processing occurs based on each user interaction event as it occurs, for each individually defined meta-data object, with any associated automated events execution as defined in the meta-data structure, and according to the standard execution order of subsequent visual structure elements. Session events such as the manual user resizing of any visual objects automatically triggers the resizing of any related visual objects based on the defined Alignment Rules for the objects.
- **User Interface Object Completion**: any defined Completion event is executed (optional) on each visual structure object before each object is terminated for the current session, although visual structure objects can be re-instantiated as requested throughout the session.

The additional logical processing and data processing functionality are progressively invoked as required by the non-visual and visual processing that is invoked within the above execution steps, based on the defined meta-data model structure.

6.2.2.3 Thin Client Options

The internet revolution has brought about massive paradigm shifts in the availability and presentation of global applications via various internet technologies, most typically presented via a web browser. Arguably these web served applications have not generally been of the rich standard historically provided by the more traditionally developed thick client graphical user applications, although the web

based user experience has been steadily improving with increasing bandwidth availability and the progressive usage and uptake of user interfaces enhanced by technologies such as JavaScript [194], Ajax [195] and more recently Adobe's Flex [196] and Microsoft's Silverlight [197] which can now approach the fuller richness of the thick client applications. These new web deployed applications are referred to as Rich Internet Applications (RIA).

However the perennial problem with browser based application deployments is that all browsers and their multiple versions support different technology mixes, to different degrees, and are not available on all computing platforms. Table 8 lists an estimated global browser usage proportion (from mid-2013) based on both desktop and mobile access (8) – it is important to note that such measurements are not exact in any way although comparable to other measurement forms and similar in their rankings, relative proportions and trends.

Browser:	**Google Chrome**	**Microsoft Internet Explorer**	**Mozilla Firefox**	**Safari**	**Android**	**Opera**
Share %:	36.3%	20.3%	16.6%	11.4%	5.0%	3.7%

Table 8 - Estimated Global Browser Share

The above table is only accurate for a short time as internet platforms, browser and usage evolves rapidly. Over recent years, the clear and continuing trends of the major browsers are:

- **Chrome**: rapidly emerged and gained market share.
- **Internet Explorer**: no longer in the majority and is continually losing market share from its previous virtual monopoly.
- **Firefox**: seemingly plateaued or slowly reducing.
- **Mobile Browsing**: rapidly escalating with the proliferation of smartphones, pads and tablets.

On further analysis of individual browser versions, which is an important issue as different browser versions of the same product have different underlying technology

support capabilities, taking the leader Internet Explorer as an example shows a breakdown in 2010 as follows (9):

Version:	IE8	IE7	IE6
Share %:	49.0%	19.3%	28.0%

Table 9 - Estimated Internet Explorer Version Proportion

Clearly there is often very large proportion of Internet Explorer users who are maintaining their use of older versions, most likely due to the need for compatibility with legacy browser based application, or an upgrade restriction imposed by Microsoft on compatibility with their underlying Microsoft Windows operating system or perhaps through a lack of active user interest or capability in upgrading.

Interesting to note is that similar comparisons with the usage of Firefox have indicated e.g. that 99% of all Firefox usage is at v3.X+ (10), possibly due to the more pro-active upgrade advice proferred by Mozilla, as well as the likelihood that installers of these non-default browsers may be more technically literate and prone to readily updating.

Another major issue lying underneath the basic browser usage statistics is the host platform as this can introduce further differentiation of capability into the same browser family. A current estimate of desktop host operating systems is provided as (11):

Operating System:	Microsoft Windows	Apple Mac OS	Linux	Other
Share %:	91.2%	4.1%	1.6%	2.4%

Table 10 - Estimated Global Client Operating System Share

Microsoft Windows continues as the dominant desktop leader. The apparent dominance of Windows is still not homogeneous with an internal breakdown of

versions as; Windows XP (23.9%), Windows Vista (3.1%), Windows 7 (52.7%) and Windows 8 (5.6%).

For the rapid growth in mobile browsing the same source provides estimate source platforms as:

Mobile Browser:	**Android**	**Safari iOS**	**Opera Mini**
Share %:	41.5%	35.6%	22.8%

Table 11 - Estimated Global Mobile Browser Share

In the mobile browser there is no clearly dominant leader and even more rapid recent evolution due to the phenomenon of smartphones plus Apple's iPad and Android devices. Again over recent years, the clear and continuing trends of the major mobile browsing platforms are:

- **Android**: upwards trend following rapid growth as it has been adopted by many equipment providers.
- **iOS**: rose rapidly and oscillating.
- **Opera**: maintained a similar level although currently trending downwards.
- **Windows Phone**: slowly increasing
- **Blackberry**: rapid escalation with a strong recent downwards trend.
- **Nokia**: trending downwards.

As there is no clearly dominant browser, nor desktop, nor device or operating system a development environment is required that provides the required rich interaction with deployment maximised across the client browser and operating system environments. The major development environments are reviewed in the following table.

Technology	**Description**
HTML5 [198]	Is the latest version of the core technology mark-up language used for common content on the world wide web. Its advantages include;

Technology	Description
	language improvements, multimedia support and interoperability with potential for cross-platform mobile application usage.
Adobe Flex (12)	Is a framework where a developer or application creates user interfaces by compiling MXML, which is an XML based interface description language, into a compressed SWF file to be executed on the client by the Adobe Flash player which is available as a plug-in on the majority of client systems.
AJAX (13)	Acronym for "asynchronous JavaScript and XML" consists of interrelated web development techniques to create rich internet applications. Using AJAX, web applications are able to asynchronously retrieve data from servers without refreshing the current page. AJAX is compatible with the majority of modern web browsers. Although supported by third party add-ons, AJAX is considered by many to offer a less rich user environment than other dedicated frameworks.
Google Web Toolkit (14)	Often referred to as GWT, is a framework aimed at Java developers to compile Java into JavaScript files to be deployed as AJAX based web applications. Compatible with JavaScript functionality of modern browsers.
Java FX (15)	Based on a new scripting language FXScript, and leverages the capabilities of the client Java runtime environment. Requires the Java Virtual Machine installed on client which is available for major operating systems.
Microsoft Silverlight (16)	Is a framework where a developer or application creates user interfaces by compiling XAML, which is an XML based interface description language, into a compressed XAP file to be executed on the client by the Silverlight plug-in which is available for major browsers on Windows and Apple Mac OS, plus some additional minor browser and platform options.

Table 12 - Leading Rich Internet Applications Development Environments

For general richness of the user experience, Adobe Flex and Microsoft Silverlight have been considered as the leaders. In terms of deployment Adobe Flex is available on more non-Windows systems than Silverlight although there are currently some major platform absences such as Apple's iPhone and iPad.

If absolute and ubiquitous market penetration and accessibility is the most important consideration then an AJAX based solution with additional functionality such as Google Web Toolkit could be considered.

HTML5 perhaps has the greatest potential as a practical standard that will be accessible across the majority of platforms.

Ultimately, many decisions will be based on the current knowledge and technology set of the developer to remain with a current or closely related technology that minimises the overall retraining and uptake effort.

Our recommendation for a thin client solution that seeks to optimise the user experience and based on deployment share would be the adoption of HTML5 based primarily on the potential market penetration. However, this also needs to be considered within the context of the other application layers of the meta-data EIS application runtime engine as discussed further in this chapter.

6.2.2.4 Thick Client Options

The traditionally developed application optimised for direct execution on a local client computer still maintains the primary benefit of a richer user interface over most of the thin client or browser based alternatives, although this lead is not as much as it has been with the advances of technologies such as Adobe's Flex and Microsoft's Silverlight.

Typically, the primary consideration for the local client hardware would have been the personal computer and it's most populous operating system, Microsoft Windows, which has long maintained a clear majority over all rivals such as Apple Mac OS and the multiple flavours of LINUX, as shown in Table 10 - Estimated Global Client Operating System Share . However, the emergence of smartphones combined with the recent and successful relaunching of tablets based on Apple iOS and Google Android are presenting a rapidly evolving mix of popular platforms. Moreover, the emerging expectation of savvy and connected users is that the same applications and data should be available and synchronised between their combination

of desktop PC, portable PC, tablet and smartphone to maximise personal connectivity and productivity.

Gartner, a worldwide technology tracker and forecaster, provide sales data showing that portable computing devices (tablets, smartphones) have long overtaken desktop computers. A snapshot of expected operating system sales for 2014 indicate the following proportions [199]:

Operating System:	**Android**	**Windows**	**Apple**	**Other**
Share %:	48.0%	13.7%	11.1%	27.1%

Table 13 - Estimated Operating System sales 2014

Noticeably, Android is the clear overall leader - although these results are clearly skewed towards the many Android and iOS consumer used devices rather than the more business oriented usage that the thesis solution would at least initially be aimed towards. However, with such a rapidly evolving consumer computing environment, a clear single option for a thick client solution is also not readily presented, although several options can be reviewed:

- **Microsoft**: as a potential single environment application. This option relies on Microsoft continuing its strategy to maintain Windows 8 and later versions on the ARM processor and thus supporting an appropriate level of Windows 8 application compatibility on all platforms, via a .NET based development environment. This would be a longer term development strategy which if successful would maximise desktop computer penetration and leverage Microsoft's successes in the mobile device market.
- **Java Client**: provides the potential to develop a single client application for Windows and Android devices. The main issue is that Java capability is delegated to the device manufacturer (17) so there can be some uncertainty around full compatibility and feature set, although for the Android platform Google as a major player is dedicated to maintaining the

Android Java environment (18). Java's availability for the iOS platform is not formally supported.

- **Multiple Client**: involves developing separate applications for each of the major computing environments. This would require separate versions for each of Windows, Android and iOS in order to capture an even higher market proportion. Unfortunately, these different development environments do not readily facilitate code migration so a high proportion of development duplication would be a requirement.
- **Cross Platform Development**: several developers are implementing development environments that are aimed at cross-compiling applications to multiple target operating systems. As these develop they may offer appropriate functionality.

The above options indicate that no single thick client development will be capable of achieving global coverage although consideration to the development of multiple thick clients can progressively increase this share. Other key issues are the:

- **Complexity and development effort of the thick client application**: is considered to be high as it is part of a full application model execution framework rather than a single purpose application. Accordingly, multiple platform developments would attract considerable proportions of additional developments.
- **Performance of the application on lower power portable devices**: has always impacted portable applications considerably and this application will be of reasonably high complexity although expected to be less so than of real time computer games and rendering which are appearing on mobile devices. The development and use of newer generation and multiple core processors for mobile devices should further alleviate processing bottlenecks.
- **Full application architecture**: whether the thick client application is just for the user interface or as part of a fully localised client installation for the entire runtime execution engine has a major impact as it is not expected that portable devices will have either the processing power nor the third party application support for other runtime engine components.

- **Intended market of the application to consumers**: the application models to be executed via the thick client application are of the Enterprise Information Systems class and as defined, are based around the entry of and access to corporate style information. Whilst this definition can be progressively extended with additional libraries to provide for general consumer graphical appeal and add further non-core functionality to extend the range of applications, it is not within the current scope to provide an execution capability for any type of application. Hence, the initial appeal is expected to be for the support of and access to corporate EIS applications rather than broad base and consumer or feature specific mini-applications.
- **Source code porting utilities between multiple development environments**: are available that can provide accelerants to multi-platform application development, and thus potentially significantly reduce the overall multi-platform development effort and cost.

While CPU power on portable devices (tablets and smartphones) has been growing rapidly, another question is how feasible would the development of a full runtime execution engine be for 100% execution on portable devices. This likely practically limits portable device availability to a standalone thick client application that provides for say the user interface alone – at least for the next few generations of device.

As a general recommendation, of thick vs thin clients, our recommendation is for a single thin client solution that provides the widest availability as discussed in 6.2.2 User Interface Elements .

6.2.3 Logical Processing Elements

The underlying business logic layer is perhaps the core element of the runtime execution engine as it manages the requested interactions from the source users and governs the required actions in terms of data transactions and workflows.

As the core requirement is a runtime execution engine to support Enterprise Information Systems there is a general expectation for scalability, performance and architectural reasons, that the logical processing elements are necessarily able to reside on separate and purpose business servers. To satisfy true scalability for the

enterprise, all elements need to be part of a fully distributed solution, ultimately to become part of a shared cloud based service.

Given the previous recommendation of a thin client solution for maximum access pervasiveness, the business logic layer of the runtime execution engine will consist of two primary logical processing elements:

- **UI Processing Service**: will collate and maintain the required user interface elements for a user session as required to define the currently displayed and active user interface objects and present to the local thin client rendering engine.
- **Logic Processing Service**: processes all requested non-locally processed user interactions or remote interactions to identify, determine and execute the modelled application functionality. Interacts with the Data Management layer via data and meta-data transaction requests.

The general implementation option for the logical processing elements are to develop a custom processing application, providing full functionality but at potentially higher cost, or to utilise a third party business rules engine, that may offer simpler development with the potential for greater flexibility.

6.2.3.1 Logical Processing Design Requirements

The core requirement of the logical processing elements design is that the required functionality is fully provided as per the design specification listed in the appendix attachment Temporal Meta-Data EIS Application Model relating to the logical processing design of the runtime engine.

The key relevant model components listed in the appendix are:

- **Logic Events**: the allowable events for object types and the defined events and conditions for all model objects in the application model.
- **Logic Functions**: all defined in-line and named functions, and their full definition and syntax, for execution by any related model objects.
- **Logic Workflow**: the application level user action workflows that are defined to provide high level interaction between users, groups and defined application objects.
- **Logic Definer Access**: models the runtime environment security permissions as they relate to allowable customisation and extension of the core application functionality.

- **Logic Variant Access**: models the runtime environment execution paths of allowed customisations for users, groups and the entire application.
- **Logic Version Control**: general template to apply to all model objects to manage versioning of core model objects including Variant Logic.

These internal design aspects govern the core definition of the application functionality and workflow as captured in the model. Other aspects that are not captured and provided as visually modelled design elements are listed in the attached full meta-data EIS application model design extraction document.

6.2.3.2 UI Processing Service

This service is only required to manage the visual interaction with users as a means to interpret user actions and responses to identify and define the appropriate core runtime command to be passed to and executed by the Logic Processing Service.

The UI Processing Service interacts with the other elements of the runtime execution engine as follows:

- **Thin Client User Interface**: via the chosen third party visual rendering APIs.
- **Local Logic Processing Service**: can interface directly to a local instance of the Logic Processing Service via its defined APIs.
- **Remote Logic Processing Service**: utilise standard web service calls to a remote or cloud instance of the Logic Processing Service (see Chapter 8 - Universal Access to Temporal Meta-Data Framework for EIS in the Cloud).

The general workflow for each user interaction of the UI Processing Service are:

- **User interaction**: is contained within the local execution environment of the thin client rendering engine providing local execution of supported user interface functionality. E.g. visual object manipulation, local validation.
- **External Event Trigger**: occurs when the local execution objects' functionality is exhausted and a defined event is passed on to the UI Processing Service by the third party visual rendering API. E.g. click on button, close window, scroll through data.

- **Form API / Web Service command**: UI Processing Service maintains a list of all current visual objects and their local state and combined with the external event trigger from the third party visual rendering API, forms and executes the appropriate API or web service command to the Logic Processing Service.
- **Interpret API / Web Service response**: based on the response from the Logic Processing Service the UI Processing Service may:
    - Do nothing. E.g. there was no defined further event action required.
    - Transform additional (visual) message to third party visual rendering API. E.g. error message or response that is additional to the current visual display.
    - Update individual visual object definitions to third party visual rendering API. E.g. return result set, changed state.
    - Update visual object list to third party visual rendering API. E.g. close window, expand displayed objects.

A separate managed instance of the UI Processing Service is required for each invoked user session.

6.2.3.3 Logic Processing Service

An instance of the Logic Processing Service can support multiple UI Processing Service instances. In a highly parallel and distributed execution environment, multiple Logic Processing Service instances can execute from a common model definition to provide execution load balancing.

This service responds to local API or remote web service commands to identify, determine and execute the modelled application functionality by maintaining the state of currently invoked objects and executing each received user interaction command against the allowable interactions for the objects as defined in the core application model definitions. Data transaction requests are forwarded to the Transaction Processing Service as required.

The Logic Processing Service interacts with the other elements of the runtime execution engine as follows:

- **Local UI Processing Service**: can interface directly to a local instance of the Logic Processing Service via its defined APIs. A thick client

instantiation for the user interface component would effectively simulate a local client version of a Local UI Processing Service, communicating via direct API calls.

- **Remote UI Processing Service**: utilise standard web service calls to a remote or cloud instance of the Logic Processing Service (see Chapter 8 - Universal Access to Temporal Meta-Data Framework for EIS in the Cloud). A thick client instantiation for the user interface component would effectively simulate a local client version of a Remote UI Processing Service, communicating via web service calls.
- **Local Transaction Processing Service**: can interface directly to a local instance of the Transaction Processing Service via its defined APIs.
- **Remote Transaction Processing Service**: utilise standard web service calls to a remote or cloud instance of the Transaction Processing Service.

The activities of the Logic Processing Service can be broadly categorised into the following main areas:

- **Service Initialisation**: the initialisation context of this service is analysed to identify and verify the status of, and to initiate any verification handshaking communications:
  - the appropriate meta-data model(s) to be processed by this service,
  - the related and authorised architectural components that can request commands, i.e. which UI Processing Services,
  - the Data Processing Services that this service will use for data transactions,
  - the eligible users that can be serviced by this service.
- **Model Initialisation**: where the meta-data model structure(s) are verified for model integrity and initial object loads of core model elements as a preliminary execution cache. Note that an internal instance of a meta-data Transaction Processing Service (as a virtual instance of the Transaction Processing Service for application data) is a requirement within the Logic Processing Service to manage access to the meta-data model structure as required.
- **Command Requests**: are synchronous tasks that occur on demand as a result of a local or remote session and are processed sequentially. They

can also include updates to the core meta-data definitions or any defined Variant Logic. The general workflow for each Command Request to the Logic Processing Service are:

- **Command Validation**: verify that the syntax, security, scope of the command is valid for the current model, instance, time, object, state, user etc.
- **Execute Command Request**: dependant on the particular command request (see Chapter 8 - Universal Access to Temporal Meta-Data Framework for EIS in the Cloud) any combination or ordering of the following outcomes may occur:
  - **Update Model State**: may require that the state of model elements is updated, or for the instantiation of new model elements as they are invoked.
  - **Execute Events and Functions**: perform the execution of any defined functions, which may invoke other visual or non-visual events.
  - **Execute Data Transactions**: invoke defined data transactions for data retrieval and/or update/creation. perform the execution of any defined functions, which may invoke other visual or non-visual events.
- **Return Result**: may be a visual result set (changed status or definition of the currently displayed visual objects), or a non-visual result such as the result and/or status of a logical execution or data transaction.

- **Asynchronous Tasks**: are internal management tasks of the service that can be further defined as:
  - **Condition Monitoring Tasks**: are tasks that need to be executed based on defined logic within the model definition. E.g. workflow conditions, watch-lists.
  - **Service Optimisation Tasks**: are tasks that the service requires to be executed on a regular basis in order to optimise the general operation and performance of the overall service execution, including the analysis of and grouping of combined visual structure elements for accelerated caching of common session

objects (could be tailored based on processing and performance capacity to pre-optimise to any combinatorial limit).

In a highly distributed execution environment multiple Logic Processing Services can be utilised to independently execute the processing of a partial load of the gross user sessions, however the core meta-data model definitions must be managed across all service instances to ensure a synchronised maintenance of and model meta-data changes.

6.2.3.4 Logical Processing Service Implementation Options

Following on from the recommendation for a thin client solution, requires a UI Processing Service that closely interacts with the chosen API option. There are many high quality third party development tools that can significantly enhance the capability of both platforms and optimise the visual rendering and communications with the end user client, whilst concentrating primary focus on the command interface requirements to the Logical Processing Service application.

Options also exist to incorporate a mainstream business rules engine into the core of the Logical Processing Service application. However, there are fundamental differences in the logical processing requirements of the temporal meta-data EIS model that need to engage beyond the simplistic rules engine capabilities of most of these products, due to the additional needs for managing the:

- highly recursive nature of the visual interface structures and the event and function based executions,
- dynamic behaviour introduced by the implementation of Logic Variants,
- temporal data and meta-data synchronisation.

The recommendation is to create a custom Logic Processing Service application based on similar core code as the custom UI Processing Service. The choice will be influenced by the available compatibility with the selected third party components used with UI Processing Service, as well as the similar issues discussed in the technology choice for the following transaction processing elements.

6.2.4 Transaction and Data Management Elements

The transaction and data management layer represents the final major interface between the runtime engine and the underlying persistent storage that drives each instance of the temporal meta-data EIS applications.

Significant advances have been developed over time to abstract transaction and data management away from the base application logic and these technologies are well suited to greatly simplify the development, integration with, and ultimately the execution of the runtime engine.

Not only can an appropriate level of abstraction and performance be provided by utilising these commonly available components but scalability is also readily availability to support advancing the distribution of the solution into a shared cloud based service. The transaction and data management layer of the runtime execution engine will consist of two closely coupled processing elements:

- **Transaction Processing**: interfacing to a dynamically configurable Object Relational Mapper (ORM) software component will effectively abstract away the nominal transactional issues and allow the runtime engine to concentrate on the data-based internal application workflow issues.
- **Database**: complete end data storage abstraction is provided by direct interfacing only between the ORM and its supported database, most likely to the many commonly available Relational DataBase Management Systems (Relational DBMS or RDBMS), and specifically avoiding the use of any RDBMA specific constructs to ensure maximum compatibility.

Within this solution context, the implementation options for the transaction and data management elements are relatively simplified, matching the choice of ORM to an existing component based on its available functionality, primarily in dynamic configurability, interface technology, and range of supported DBMS.

6.2.4.1 Transaction and Data Management Design Requirements

The fundamental transaction processing requirements of the meta-data EIS application model can be readily abstracted as common data transactions based on the configurations specified by the state of the current meta-data EIS application model. These transactions can be easily rendered into a portable definition such as XML exist by the runtime engine for execution by third party transaction processing environments.

The key internal model structures from which the transaction processing requirements will be abstracted to are provided as per the design specification listed in

the appendix attachment Temporal Meta-Data EIS Application Model relating to the visual presentation design of the runtime engine.

The key relevant model components listed in the appendix attachment are:

- **Generic Temporal Meta Data Entity Schema**: general template to apply to all model objects to manage versioning in order to provide temporal execution capability.
- **PKFK Inheritance for Entities**: logical inheritance structures to define primary dependency model objects (only required due to some modelling limitations of the design tool used to model the temporal meta-data EIS model structures).
- **All Entities Basic Inheritance**: logical inheritance structures to fully define all required model objects.
- **Data Model Definition**: structure representation of the modelled database definitions including the additional transactional logic representations of the modelled applications.

The remainder of the internal design aspects that are not captured and provided as visually modelled design elements are listed in the attached full meta-data EIS application model design extraction document which also details the associated model business rules, parameters and options.

6.2.4.2 Transaction Processing Options

The temporal meta-data EIS model contains representations of all of the core data structures as well as all of the use cases for each data object in the specific application model.

Based on this model, every potential data transaction for the modelled application can be fully identified either in advance or on as required basis. For a traditionally developed application this may involve hundreds or even thousands of individually coded transactions, often referred to as CRUD, representing the common requirements for transactions to; Create, Update and Delete data.

As every transaction can itself be readily defined based on queries to the meta-data structure then such transaction requirements can be extracted in a portable format such as XML. The set of all such XML fragments will be used as the source definitions of the data structure and transaction requirements to the selected Object

Relational Mapper which will then manage all ongoing data access and transactions. The use of available ORM components greatly simplifies the data access issues.

The key requirements for the selected ORM are:

- an existing software component in common use and readily supported,
- dynamically configured, via XML or similar command interface,
- supports a wide range of commonly used DBMS.

The above requirement for dynamic configuration is key for the meta-data EIS application runtime engine as it needs to be able to flexibly interface with dynamic schema. The end database schema structure can be modified at any time via meta-data changes whether core or as Variant Logic, the later in particular could be a very regular occurrence. Many ORMs are restricted to manual configuration via their user interface as part of an internal mapping class generation process – for the runtime engine this needs to be able to be specified or performed automatically on an ad hoc basis to respond to dynamic meta-data changes.

Of those ORMs that most readily provide for dynamic configuration, the most suitable that address the above criteria are:

ORM	Platform	Supported Databases
nHibernate	.NET	DB2, Firebird, MS Access, MySQL, Oracle, PostgreSQL, SQL Server, SQLite
DataObjects.NET	.NET	Firebird, MySQL, PostgreSQL, SQL Server
Hibernate	Java	Extensive
Eclipselink	Java	Oracle, PostgreSQL, Sybase

Table 14 - Recommended Candidate ORM Component

Depending on the desired platform technology, Hibernate for Java, or its ported sibling nHibernate for .NET offer high levels of support, functionality, with the greatest range of supported end database systems.

6.2.4.3 Database Options

By relying on the ORM to fully manage all data transactions and interface to the end data storage, the selection of a DBMS system becomes largely transparent as most ORM systems support a wide range of DBMS.

An additional benefit of utilising the ORM is that the ORM can readily manage data that is distributed between multiple discrete types of DBMS.

The choice of DBMS is dependent on many intra organisational factors, however the selection of an ORM with wide ranging DBMS compatibility such as recommended from Table 14 - Recommended Candidate ORM Component maximises DBMS compatibility and makes the choice a trivial issue.

6.2.4.4 Physical Data Structures and External Data Access

Users of temporal meta-data EIS applications will typically only be aware of the abstracted data objects that they interact with via the runtime engine's user interface. Depending on the level of multiple aliasing that can occur, which provides an unlimited capability to privately re-define any existing data objects in a localised format, the level of abstraction from the originating source can become significantly obfuscated from its origins.

Even for advanced users that have the access to the model's closest abstraction level from the source data, the model may have incorporated preliminary abstraction to prepare, format and filter the original source data formats. In any case, even the apparently simplest of model data objects will require that source data tables will be physically structured very differently based on the following:

- The source database naming conventions may not be compatible with the generally more readable model naming conventions so will be named differently,
- Formats of the source database columns may vary in compatibility and structure from the model formats,
- The model relationships may differ from any implied relationships of the source data,
- For writable data source structures, significant additional model-control data columns may be added to the basic source data structure in order to support the higher level functionality provided by the runtime engine,

such as temporal execution capabilities and multiple lingual representations.

While technical database administrators would be able to map the required source database columns any direct data interactions on the source database columns would typically not be recommended as:

- Any changes to the source database columns would bypass the business logic of the temporal meta-data EIS application model and potentially introduce invalid data that may violate the intended business logic,
- Any changes to the additional model-control data columns would bypass the internal logic of the temporal meta-data EIS application model and invalidate the model's expected states and thus violate the overall model integrity with potentially catastrophic results,
- Even direct reading of the source database columns may not fully reveal; the correct version of the data, the desired format of the data, or the current state of the data; which are managed by the runtime engine in conjunction with the associated additional model-control data columns.

In order to correctly process, format, present and interact with the source database columns, two forms of interaction are available via the runtime engine:

- Non-updatable database views can be created by the transaction processing service ORM based on the defined and modelled business rules, for use by the target DBMS to provide read only data access,
- Full temporal meta-data EIS model command capability is provided via the web service commands detailed in Chapter 8 - Universal Access to Temporal Meta-Data Framework for EIS in the Cloud, a subset of which includes secure read and write access capability with the source data via the model's business logic.

Direct access to raw source data is rarely recommended for any controlled data environment due to the high potential to bypass and violate any data integrity and business logic rules that may exist higher in the managed knowledge environment.

6.2.5 Access Security and Requirements

Data access and application functionality security is a fundamental requirement of EIS applications. A key benefit of the meta-data EIS application model is the complete integration between the data and the modelled application logic as each

aspect; application functionality, security perspective, and secured data; is defined by (meta-)data rather than any fixed code which provides the ultimate flexibility as well as automating the security application process – once the security needs are defined, they are immediately implemented.

Access security will effectively be implemented as a core front-end module within the Logic Processing Service (see 6.2.3.3) as the processing of every command will necessarily be subject to security analysis:

- **Execution Security**: session based security to ensure the ongoing validation of user credentials to maintain continuous interaction or access to suspended sessions.
- **Access Security**: subsequent to validated execution security, the requested model command and access to processed model elements and data is managed by the security module based on the modelled security meta-data.

The standard security functionality is provided by the runtime engine but implemented based on the meta-data defined relationships between the model elements and the source data module with no further coding required.

The core requirement of the access security elements design is that the required functionality is fully provided as per the design specification listed in the appendix attachment Temporal Meta-Data EIS Application Model relating to the access security design of the runtime engine.

The key relevant model components listed in the appendix attachment are:

- **Security Execution Access**: the security access and execution authorisation hierarchies for all model objects based on defined roles to apply to identified users.

These internal design aspects govern the core definition of the security as captured in the model. Other aspects that are not captured and provided as visually modelled design elements are listed in the attached full meta-data EIS application model design extraction document.

6.3 Advanced Runtime Features

The previous fundamental requirements list the common features of EIS style applications that need to be supported by the temporal meta-data EIS runtime engine,

notwithstanding the expectations that even this basic approach offers substantial benefits over traditionally developed EIS applications.

The additional advanced functionality described in this section provide further and quite distinct advantages over what can readily and easily be achieved and duplicated for traditionally developed EIS applications. As supported functionality of a runtime engine, every meta-data EIS application is provided with these advanced features such as:

- **Temporal Execution**: maintains a synchronisation between all data changes as well as the application defining meta-data so that any version of a meta-data EIS application can be executed at any time, and with the exact data configuration at that point in time.
- **Variant Logic**: managing the execution of parallel and alternate logic to that of the core meta-data logic where the additional logic variants have been defined to supplement the core logic.
- **Automated Version Management**: every meta-data EIS application change is an individual update to the meta-data and can be tracked, synchronised and grouped. Batches of meta-data changes can be applied automatically to a user system without user involvement, including detecting any potential logic collisions between core application logic and any defined Variant Logic.
- **Data Condition Monitoring**: executing any user defined data monitoring conditions to execute user or administrator defined functions for discrete monitoring, user workflows or as part of defined Variant Logic.
- **Distributed Data Management**: where a particular meta-data EIS application is executed as part of a de-centralised execution environment of similar core applications, key data can be automatically authorised or rolled up via a defined authorisation hierarchy.

All advanced model elements of the meta-data EIS application model must be directly supported by the runtime engine as described in Chapter 5 - Instant Interaction EIS System Modeller. Further model definitions are provided as the function definitions (see Chapter 8 - Universal Access to Temporal Meta-Data Framework for EIS in the Cloud) with the detailed model designs listed in the thesis appendix attachment which provide the full specification of the design requirements.

These additional design elements for the runtime engine in this chapter focus on the additional major architectural considerations that must be supported in addition to each atomic element's specific functionality.

6.3.1 Temporal Execution

The design aspects required to support temporal execution are very similar to those provided by fully featured audit tracking functions. By also applying the well understood temporal data management structures to the structures of the meta-data, which provide the application logic functionality, then a perfect temporal synchronisation can be achieved that readily allows the execution of the correct version or state of the application at any point of time, addressing the exact state of the database at that time.

6.3.1.1 Temporal Data and Meta-Data Structures

In order to temporally manage the meta-data and data, every meta-data object and database table must enhanced with the supporting temporal structures as described in 5.2.1 Generic Distributed Temporal Meta-Data Inheritance and 5.2.2 Generic Distributed Temporal Data Inheritance respectively.

Where an end user data table may not be able to be modified to support the temporal structure, a parallel and separate structure can be established to provide the managing temporal versioning and audit data attributes.

Particularly for temporal data, due to the potential net volume of updates to individual records, each of which triggers a new temporal record copy, it is recommended for performance reasons that separate data table structures are maintained for each temporal data table as:

- **Current Data**: only the single latest version of each data record is maintained to optimise database runtime access for all current time transactions, with each data table still modified as temporal data,
- **Full Temporal History**: utilise a separate set of tables for each data table, all modified as temporal data and containing all temporal data records, which are continuously updated as each data transaction occurs and are used as the exclusive source for any specific temporal transactions.

Temporal meta-data can also be optionally separated into separate meta-data repositories following a similar reasoning to the above temporal data separation. This

option would be recommended if there was an obvious performance impact caused by maintaining a merged temporal meta-data structure – this is a particular risk where dynamic retrieval of the meta-data occurs on demand as required from within the Logic Processing Service (see 6.2.3.3).

6.3.1.2 Temporal Data and Meta-Data Update

A similar algorithm is implemented to manage both temporal data and temporal meta-data model definitions, in whatever format the latter is ultimately stored and managed in.

The relatively simple steps for a full temporal repository are:

- **New Record**: insert new record with a Temporal Identifier Timestamp equal to the date / time of creation, and a Temporal Record Status of Current, and apply the core data to be saved.
- **Updated Record**: locate the current record as identified with a Temporal Record Status of Current, and change to a Temporal Record Status of Superseded. Create a new record as described above, copying any existing unchanged core data and updating the changed core data.
- **Deleted Record**: locate the current record as identified with a Temporal Record Status of Current, and change to a Temporal Record Status of Superseded. Create a new record as described above, copying all existing core data, and change to a Temporal Record Status of Deleted.

Where a copy of only current data and/or meta-data is to be maintained separately for performance optimisation from the full temporal repository the following additional steps are required to maintain the minimised current repository:

- **New Record**: insert new record and apply the core data to be saved.
- **Updated Record**: locate the current record and update the changed core data.
- **Deleted Record**: locate the current record and delete it.

Clearly, the above steps for the minimalised current repository are similar to the basic CRUD operations that are performed on common database tables, however there are also additional non-temporal data attributes to be maintained by the runtime engine for each data / meta-data table to support other advanced features of the temporal meta-data EIS application framework.

6.3.1.3 Temporal Execution

A major aspect of implementing full temporal execution capability is provided as a temporal selector in the user interface to an authorised user. The temporal selector simply permits the selection of the required date and time for temporal execution.

The temporal meta-data EIS application is then executed with the exact application state and logic as it was on that selected date / time, by applying the exact meta-data model as it then was. Similarly, the exact data set and state at that time is also then presented to the application.

The fundamental temporal selector is in the form of:

- a date / time control to specify the required temporal execution commencement, with options to save a temporal starting point as a named point in time for further reuse, including specifying an optional temporal analysis end point, plus managing multiple timelines into common hierarchical groupings, and
- a selector navigator for any of the above defined timeline markers.

Once a temporal selection has been made and the user's session is now operating in temporal execution mode there are two key principles governing the runtime engine execution for this specific session:

- The internal data transaction views that normally govern data and meta-data retrieval, are dynamically modified to include additional filtering and source identification to respectively apply the current temporal execution date / time, and the source location of the full temporal repository database tables.
- Any data change transactions are restricted to read only as regards to the primary database, or otherwise limited to interim operations on temporary data. i.e. temporal execution is to be provided as a read only capability in order to maintain data integrity. This can be readily applied by the runtime engine as a read only attribute mask over existing user security access and transaction execution.

During temporal execution, any normal authorised application feature that the user would then have had available, according to the security provisions in force at that date / time, are made available, and the application executes exactly as it did then and with the exact data set as was then available.

6.3.1.4 Temporal Rollback and Rollforward

In addition to temporal execution of temporal meta-data EIS applications at a specific date / time, the runtime engine can support incremental deviations in time to facilitate the forensic analysis and effect of individual or batched transaction changes on the source database. These are referred to as:

- **Temporal Rollforward**: is enacted as an increase in the currently selected temporal execution time,
- **Temporal Rollback**: is a decrease to the currently selected temporal execution time.

It is important to note that these progressions through time are not enacted as a re-application or undo of transactions that occurred within the deviated time period, but are simply a re-definition of a new temporal date / time to be applied as the new current temporal selection.

The Temporal Rollforward or Rollback functions can be operated in a variety of forms by:

- **Absolute Temporal Selection**: is purely time based by specifying an absolute date / time point by:
  - selecting a new date / time, or
  - selecting a (pre-)defined temporal marker to progress to.
- **Relative Temporal Selection:** is event or transaction based by selecting a specific data or meta-data transaction that occurred on a relative basis from the current date / time. This may be implemented in a variety of forms that step through each successive or previous data or meta-data change that occurred on an individual or batch basis, which can be performed and facilitated by mechanisms such as:
  - Click on directional selectors that will increment (or decrement) to the next (or previous) implemented transaction,
  - The above directional selectors can also be supplemented with batch icons that can change to the +/- Nth next (or previous) implemented transaction,
  - Provide differentiation to the directional selectors that can select between data or meta-data changes,

- Provide a configurable ordered list of data and/or meta-data changes that can be used to more clearly identify the scope of each change and then select the transaction from that list.

When the Temporal Rollforward or Rollback event involves only data changes then there is generally no impact on the currently displayed visual elements other than as dictated by the underlying logic of the meta-data model – rather the display of data may be refreshed as required. The primary exceptions that would occur are examples similar to where child data was being displayed as visual elements while the next transaction then deleted the related parent (and child) data resulting in an empty child display that normally would not have been possible according to the application logic – these examples need to be monitored by the runtime engine to preferably identify the 'loss' of the displayable data and then dynamically modify the application logic to prevent any illegal operations on the then null data.

However, when the Temporal Rollforward or Rollback event involves a meta-data change there may be obvious changes to the display of the current visual elements if the meta-data change had affected any of the currently displayed visual elements. The runtime engine needs to monitor whether any visual changes are required and invoke the visual refresh as required.

6.3.1.5 Temporal Analysis and Temporal Session Update

Temporal execution combined with Temporal Rollforward or Rollback provide the ability to review previous states of the application and database, and execute all authorised aspects of the meta-data application on a read only basis i.e. no changes to the end database are made, other than any purely temporary data updates that normally occur on the basis of standard transaction processing. These features provide the temporal analysis capability.

The runtime engine for the temporal meta-data EIS application can provide more advanced temporal functionality by permitting authorised users to operate the meta-data application in a simulated interactive operational basis whereby new changes to the end database are permitted to be made on a personalised temporary basis – I refer to this as a temporal session update.

This temporal session update capability does not affect the core end database that is accessed by the general user population. Any updates made as part of a temporal session update are managed separately from the core database and are only accessible

by the defining user, although it is expected that the data associated with the temporal session updates is stored with the standard temporally managed end database.

Temporal session updates are managed as:

- The authorised user invokes a temporal session and selects a date / time based on creating or selecting a (pre-)defined timeline marker that permits temporal session update to occur. The user is then able to operate in "temporal session update" mode which replaces the usual read-only operating mode,
- The user is then able to make any data changes or transactions that would normally be permitted, although any data changes are now occurring under the management of "temporal session update" mode,
- As the advancement of time does not occur naturally whilst undergoing temporal execution, the definition of the actual transaction time to be applied for each transaction during "temporal session update" mode can be applied in either of two methods, as desired at any time:
    - **Discrete Date / Time Mode**: the temporal date / time is selected by the available time selector controls. This is adequate mode where there is only a small number of additional transactions to be made, or when there is only a relatively small temporal window of availability for the new transactions to be implemented. It can be assisted by auto incrementing by a set amount after each new transaction is executed to ensure the execution integrity of the transactions.
    - **Natural Time Mode**: once the temporal date / time is selected by the available time selector controls, normal elapsed time increments are automatically managed. This is an useful mode when there are a lot of transactions to be made, and an adequate temporal window of availability while the new transactions are implemented.

The mechanisms that support the management of temporal session updates in the temporal meta-data EIS application framework are:

- No core database data is ever changed as a result of any temporal session updates,

- Additional Temporal Change attributes are available for each data record to identify copies of real data records that are made to represent the data changes made during temporal session updates i.e. if a real record A is changed due to a temporal session update then:
    - A copy of the record is first made to record B,
    - Record B is flagged with the additional Temporal Change attributes to identify the user and temporal session timeline marker (which then apply to all subsequent updates),
    - Record B additional Temporal Change attributes are flagged as the Current record as per usual temporal data management rules,
    - Any subsequent changes to the "original" record are now applied to Record B and its new line of temporally managed records according to the additional Temporal Change attributes.
- Any subsequent data retrievals that return instances similar to Record A above, i.e. that have now had temporal session updates applied, must then invoke a secondary nested retrieval to return the correct temporal version from the new line of temporally managed records according to the additional Temporal Change attributes, for each "original" record.

The above mechanisms are dynamically embedded within the temporal database views for all database transactions whenever temporal session updates have been invoked.

This temporal session update capability significantly enhances the temporal execution capability by providing a temporal "what-if" capability based on the original temporal data.

6.3.2 Deployment and Execution

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 application update process for meta-data EIS applications can be greatly simplified for both developers and end users. We can remove the need for specific version upgrade programs and procedures to be created by application developers 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 update process as all updates can be applied sequentially and as a single process rather than as multiple separate upgrades.

End user organisations need only to manage the availability of a common local runtime environment for the meta-data EIS framework, and to manage the local users' access. As application updates are made available by the developers, the update meta-data streams can be deployed locally according to their local management policies, and ultimately by authorising the local runtime environment to process the selected meta-data changes.

This standardised update process can also provide a clear identification of all application changes that will be made to the end-users, optionally before any update is to occur, by reporting on the exact nature of the meta-data changes and clearly identifying all changes and any associated impacts. Such a precise identification of all changes including any potential logic conflicts with existing Variant Logic can be used to target specific training needs as well as provide advance warning of any re-engineering requirements for any conflicting Logic Variants.

Once the meta-data EIS application is deployed to an operating instance, as a combination of the runtime execution environment populated with any number of meta-data models as the individual meta-data EIS applications, each instance will execute according to the defined meta-data model and users will operate within the locally defined security roles.

An additional execution option called Data Condition Based Monitoring can be defined locally using the inherent event definition capabilities of the meta-data EIS framework to set local conditions to provide for; monitoring data thresholds, statistical analysis, security watch processes, and triggering additional user defined data processing operations beyond the core application functionality.

The above aspects of meta-data EIS application execution are reviewed in further detail in the following sections.

6.3.2.1 Automatic Application Update

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, as per 4.8.3.1 Automatic Application Meta-Data Version Control.

To perform the meta-data update is a two-step process:

- **Defining the Meta-Data Update**: identifying the scope of the meta-data update i.e. performing partial incremental meta-data updates where there are a series of candidate major meta-data updates available.
- **Automated Meta-Data Update and User Customisation Detection**: performing the actual or simulated update of the selected meta-data changes, including identification of any instances where defined Variant Logic may now be in conflict with the new meta-data logic.

The above aspects of the meta-data update are reviewed in further detail in the following sections.

6.3.2.1.1 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.

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 47);

- **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.

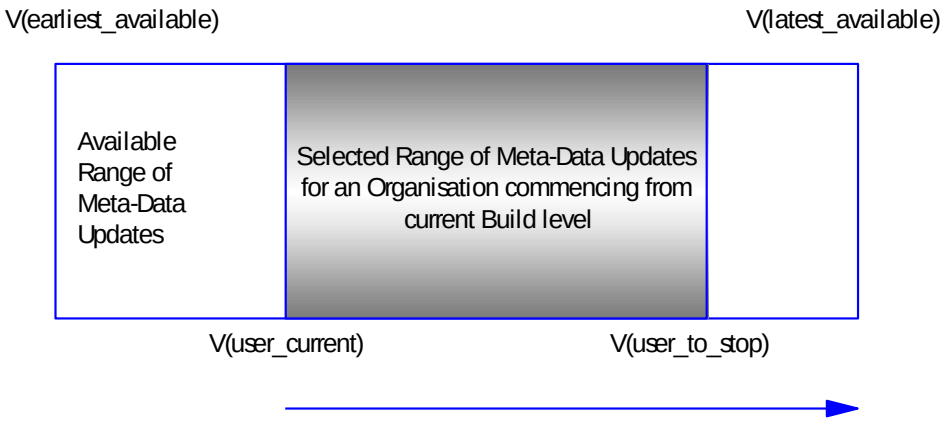


Figure 47 – Optional range of selected meta-data update

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 48).

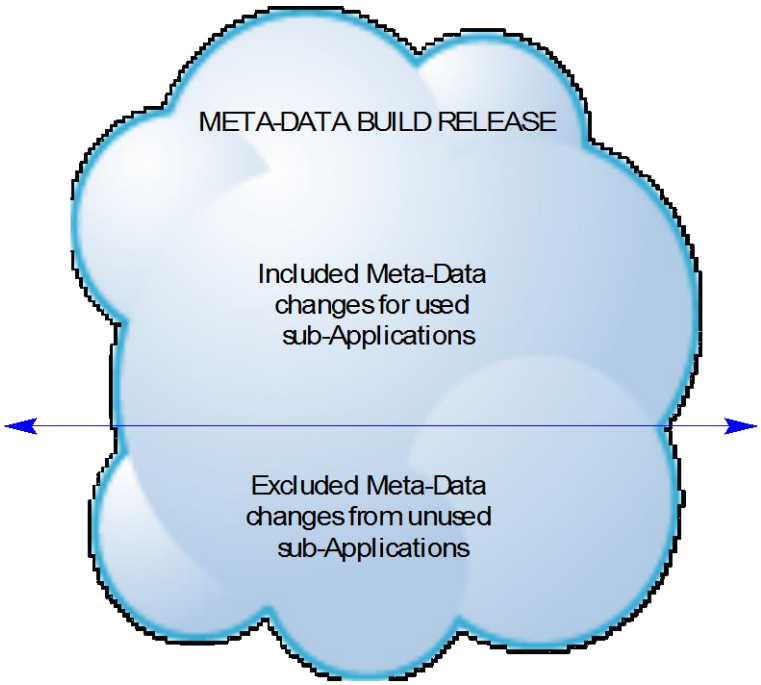


Figure 48 – Optional scope restricted build for a meta-data update

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.

6.3.2.1.2 Automated Meta-Data Update and User Customisation Detection

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.

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 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.

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 organisations.

6.3.2.2 Temporal Data and Application Snapshots

The only requirements to execute meta-data EIS applications are to establish a runtime execution environment and to then apply an appropriate stream of meta-data to represent the application logic. An associated option with the meta-data stream can be a data stream which can be used to pre-populate the defined data structures according to the meta-data logic.

This accompanying data stream, when extracted from a live or production meta-data EIS application, can facilitate the creation of working or archival snapshots of the meta-data EIS application, also with optional full temporal data execution capabilities.

There are two key aspects of the snapshot to be defined for the meta-data and data extraction:

- **Temporal Data Period**: what subset of the available temporal data will be extracted can be defined as:
    - **Temporal Period**: a selected time period where meta-data and data is provided to cover all existing data and transactions within the selected time period. Full temporal execution is available within the selected time period.
    - **Temporal Point**: no temporal data is included, only the exact meta-data and data as defined at the selected point in time. No inter-temporal operation is available.
- **Execution Capability**: defines the limitations that will apply to the ongoing execution of the snapshot instance, in addition to but over-riding the standard security definitions:
    - **Read Only**: the snapshot can only be used to execute the application in a read only mode, no data can be changed. Useful for providing an offline or portable instance of the meta-data application for analysis or audit.
    - **Write Data**: the application can be fully executed within the normal security role permissions, data can be changed or added although no application logic can be altered. Useful for establishing as user test or training environments.
    - **Variant Logic**: both meta-data and data can be changed or added within the normal security role permissions. Useful for providing standalone meta-data development environments.

The use of the snapshot feature can readily facilitate a variety of alternate execution environments to support the common needs of; operational data archival, executable audit, historical reporting and analysis, feature testing, user training, meta-data application development.

Armed with a common runtime execution environment with a supported back end database, snapshots allow unlimited access to multiple meta-data EIS applications, versions and instances.

6.3.2.3 Data Condition Based Monitoring

Traditional application systems that need to invoke ad hoc monitoring of application events generally need to program the logic into the specific application objects as customisations, or may provide limited monitoring capability via pre-programmed features. At the database layer the use of database triggers has long been a useful means of inserting ad hoc or even standard data processing logic into the data or application environment.

Significantly enhanced condition based monitoring and processing capabilities can be provided by the meta-data EIS application via its event processing capabilities, see 5.3.3.3 Visual Structure Element Events and 5.4 Program Flow Elements .

These event processing definitions can be applied to any defined meta-data object as well as any data object. Combined with the range of possible root triggering events, this provides an unlimited capability for authorised end users, analysts, system administrators and security specialists to define their own purpose defined events to provide condition based monitoring over any authorised combination of users, data, workflow, events and processing logic.

This out of the box functionality is implemented via the inherent Variant Logic capability and authorisation structures and can be readily used to provide additional operational application execution features such as:

- **Monitoring** thresholds of data values to provide advance or forecast warnings,
- Provide statistical **analysis** of application segment or object usage,
- Establish seamless **security** watch processes triggered on specific data or user access,
- Trigger additional data **processing** or notification operations beyond the core application functionality.

The range of logical extension is virtually limitless as it is supported by the full range of available Variant Logic definition, and can also invoke the standard workflow functionality to incorporate the condition based monitoring logic into other pre-defined organisational procedures.

6.4 Conclusion

The design overview for a prototype runtime execution engine described in this chapter addresses the major conceptual design models presented in Chapter 5 - Instant Interaction EIS System Modeller clarifying an overall execution architecture for the meta-data EIS applications that can also support a maximal mobile user base via global cloud based services.

The key to maximising global and ubiquitous access to applications via the cloud is twofold. Firstly, to present a portable user interface that can reach out to the maximum number of potential users when real-time user interaction is required, and secondly, to provide a standardised and secure access to all of the system's features that can be addressed between systems to provide direct and automated interfacing.

Whilst current technology, and more importantly, the business world of competing technology vendors, does not yet fully permit any guaranteed portable model of execution, particularly working downwards from the upper application layers, our analysis of available and trending technologies point the way with a strategy to readily achieving a maximal user base penetration.

Beyond satisfying the core requirements for general model based execution, the advanced features of the meta-data EIS application model framework and their associated execution requirements have also been clearly expanded from the base design models.

The stated architecture will also utilise secure web services as the inter-module and inter-instance interface standard, permitting alternate platform modules as required, and supporting global cloud access to secured model objects wherever they are available and required. The detailed syntax for this access is expanded in Chapter 8 - Universal Access to Temporal Meta-Data Framework for EIS in the Cloud.

In this chapter, I have described the design of a prototype architecture and runtime execution engine model for the meta-data EIS framework that can implement the modelled features. The following chapters will expand on this detail to more fully specify how models can be created and used as working meta-data EIS applications.

Chapter 7 - Accelerants for the Iterative Design of EIS Models

7.1 Introduction

The definition and subsequent execution of the core model structure for the meta-data EIS applications has been the primary focus of the previous chapters. This key preparatory issue as to how to efficiently define the meta-data that would in traditional systems be coded by computer programmers.

The history, complexity and scope of large scale EIS applications tend to have resulted in large code bases that often include a multitude of disparate technologies and coding systems. Components of the runtime execution engine for the meta-data EIS application framework are also likely to potentially consist of a variety of technologies, however the key issue is that these components need only be developed once and then globally used to execute any and all of the defined meta-data EIS application models, not unlike common library environments that support Java and .Net, although providing much higher level application functionality.

One of the key objectives of the meta-data EIS application framework is to shift the application development requirement for the EIS application logic from programmers to application users, with the greater business logic complexity

emphasis on power users and business analysts. This greatly expands the accessibility of both original application defining and any subsequent local application modifications that I term Variant Logic.

All of the basic application development analogies still exist for the meta-data modelling processes along with some unique aspects as facilitated by the use of common meta-data modelling. Defining application model meta-data broadly falls into a combination of the following:

- **Defining new meta-data**: creating new meta-data definitions for the modelled application,
- **Deriving the meta-data**: from some existing non meta-data EIS application based objects such as reverse engineering from existing database schemas,
- **Editing existing meta-data**: to modify existing aspects of a model as core application logic or extended Variant Logic,
- **Merging meta-data models**: where multiple meta-data EIS application models exist their meta-data models and thus functionality can be merged.

As the meta-data model is fundamentally structured data, the meta-data can even be hand loaded into the appropriate data structures for subsequent execution however this is hardly a user-friendly option. To efficiently manage the creation and maintenance of the meta-data models requires assistance editor software analogous to the Integrated Development Environments (IDE) of traditional software development. In some areas this editor would be simpler than common IDSs but also necessarily include additional higher level features that need to be modelled as part of the meta-data EIS framework which may again be analogous to the various component add-ons that are available to common IDEs.

The sections in this chapter describe the general operation and requirements for effectively defining the model meta-data as defined in Chapter 5 - Instant Interaction EIS System Modeller for execution by the runtime engine described in Chapter 6 - Agile Platform for Dynamic Systems Change Management.

7.2 Integrated Meta-Data Modelling Environment Editor

When considering the structure of the meta-data EIS framework model, the key issue is to efficiently populate the model structure with the appropriate meta-data to

represent the desired application model. Clearly some form of a GUI based meta-data editor is required as a key component in the application modelling process. I denote the collective meta-data definition functionality as an Integrated Meta-Data Modelling Environment (IMDME).

At the broad level. when considering the common application layers of an application's user interface, application logic and data layers the common visual IDE metaphors of a GUI form designer, workflow designer and data modeller working across an integrated model structure still seem to be an adequate basis of an appropriate IMDME solution.

The primary difference between traditional coding based IDEs and the IMDME are that an IMDME:

- Does not need a compiler nor needs to compile (although some optional function syntax processing may be invoked to provide a level of pre-execution optimisation for function definitions, or indeed a dynamically managed compilation or runtime optimisation service could be managed automatically by the supporting services),
- Has a pre-defined model structure as the primary objective output,
- Functionally addresses every layer of an application through its defined structure,
- Maintains overall logical continuity by maintaining directly modelled relationships between every model object,
- Can offer significant options for providing "wizard based" semantic meta-data definition optimisations due to the underlying meta-data structures, to solve or resolve specific application functionality requirements.

When considering the definition of the EIS style applications that the meta-data EIS application framework proposes to resolve, and that the underlying meta-data definitions that need to be populated in order to define and subsequently execute a meta-data EIS application, it becomes clear that many aspects of an IMDME editor application are indeed similar to examples of executing meta-data EIS applications.

It is with this concept in mind that the following development options for an IMDME are presented:

- **Traditional IMDME Editor Development**: uses traditional coding based processes and solutions to develop an IMDME to suit the meta-data model,
- **Hand Code IMDME as a Meta-Data EIS Application**: avoid specifically coding an entire IMDME editor application, and rather manually populate an equivalent meta-data EIS application model with the appropriate meta-data that models the required functionality of an IMDME where the functionality is supported by the meta-data EIS runtime framework.
- **Iterative IMDME as a Meta-Data EIS Application**: similar to the above option however it minimises the initial and subsequent manual meta-data definition by progressively using the current scope and functionality of the IMDME editor to define the next iteration of functionality to be available in the next version of the IMDME.

The runtime engine for the meta-data EIS application framework must always be a new software development as described in Chapter 6 - Agile Platform for Dynamic Systems Change Management, and is clearly a pre-requisite for pursuing either of the meta-data definition based IMDME options listed above, as once developed, then any meta-data model can be executed whether the meta-data has been defined manually or from a well-defined IMDME application.

Clearly, the latter options above provide the potential for minimising the effort on coding a separate IMDME application and there is a certain elegance to pursuing the latter iterative option to progressively model and enhance the IMDME functionality within the previous IMDME application version.

7.2.1 Traditional IMDME Editor Development

The modern GUI based IDE for traditional code definition offers a very user friendly and syntax aware programming environment supporting a mix of visual tools and direct code manipulation. Most of the current common IDE features, streamlined over decades of refinement have useful analogies for IMDME editor development such as:

- IDE form designers can be readily adapted to model and display the visual structure elements for data entry and manipulation, and for data extraction and reporting,

- The hierarchical object or solution navigation browsers of IDEs can navigate various views of the meta-data model's object relationships,
- Object or class inspectors based on the meta-data model's object's properties can provide a similar secure property editing feature,
- Scripting engines can be modelled with the lexicon of the meta-data function structure to support the editing, syntax management and optimisation of all user defined functions,
- Generic object relationship diagram mappers can be adapted to manage workflow sequence logic as well as modelling data relationships.

For all of the above features there are significant existing third party toolsets in a variety of language frameworks that can be readily obtained to minimise the overall IMDME editor development effort.

7.2.2 Hand Code IMDME as a Meta-Data EIS Application

An IMDME editor needs to prompt for the entry of meta-data EIS model meta-data and store the meta-data as data in the appropriate model data structures. An interesting example of an IMDME editor could an instance of a meta-data EIS application itself.

Our original definition of EIS applications as defined in 4.2.4 Enterprise Information Systems Definition summarised them as "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 to process the data, and utilise database transactions".

The primary implication for user interface definition is that other than promoting efficient and effective use for the transactional data entry and manipulation, the user interface would not normally be expected to require any extraordinary user interface artefacts other than as typically found in the common suite of user interface object types found in most GUI environments.

This implied restriction, which is only limited to the current minimal set of user interface objects in this initial version of the meta-data definition and can be readily extended as required, would place some limitations on the elegance and efficiency of defining an IMDME editor based solely on the current functionality that could be provided as a meta-data EIS application.

7.2.3 Hybrid Meta-Data IMDME Editor

As discussed in the previous sections, the IMDME editor would like most modern IDEs benefit from additional GUI constructs as listed previously such as form designers and object relationship modellers which are not currently required nor defined in this initial version of the meta-data EIS model as described by this thesis, although can be readily provided by:

- Extending the scope of the modelled and supported user interface controls as natural extensions of the meta-data EIS application framework definitions and provided with runtime engine execution support,
- Interfacing to third party tools that provide the required functionality via the native web services commands directly to the third party tools or via a web services wrapping layer,
- Incorporating third party meta-data EIS application segments that have been defined with specific desired functionality to readily merge with or be accessed by the meta-data EIS framework.

Any or a combination of the above methods will readily add new capability to the available functionality of the meta-data EIS framework and the resultant executed applications.

Notwithstanding the benefits of additional GUI model designer elements there are many features of an IMDME editor that can be usefully defined as meta-data EIS application segments that can continue to reduce the overall development effort in a hybrid style editor application as per the expected benefits of any meta-data EIS application, such as:

- Overall application architecture of the meta-data EIS application framework,
- Navigation, toolbar and menu controls structure,
- Hierarchical object browsing and solution navigation to navigate various views of the meta-data model's object relationships,
- Standard forms and user interface objects to act for as object or class inspectors based for secure property editing of the meta-data model's object's properties,
- Auto generation of the IMDME editor documentation and online help.

A combined approach to develop such a hybrid style IMDME editor can combine the best of third party developments with the meta-data EIS framework to produce an IMDME editor application itself which will enjoy the majority of the lifecycle benefits already reviewed in 4.9 Temporal Meta-Data Methodology for the Meta-Data Based Application System Lifecycle.

7.2.4 Iterative IMDME as a Meta-Data EIS Application

An alternate but complementary approach to creating the IMDME editor is to employ an evolutionary approach to develop a basic IMDME editor (as an example of 7.2.2 Hand Code IMDME as a Meta-Data EIS Application) and then progressively improving the functionality available from the IMDME editor's meta-data by using the newly available editor functions to define and create the new IMDME editor meta-data and thus the new functionality. In reality this would best be achieved by creating an iterative IMDME editor as a hybrid style as described in the previous section.

The accelerant key to successfully evolving the IMDME editor is not to just add new functionality to the next version of the IMDME editor meta-data that provides new single purpose features (although this is of course continually required), but to provide exponential style new functionality that can be used to then define multiple sets of features or to further facilitate new exponential style functionality.

An example of adding a new single purpose feature might be to define a new form or wizard user interface object that would be used to define the menu or toolbar navigation control structures. Whilst this is an essential and useful component of the overall IMDME editor, and must be defined at some point, it basically has only one purpose throughout the definition of any application's meta-data.

However, exponential style new functionality can be used to create or define multiple examples of new single purpose features or facilitate new exponential style functionality. An example of adding new exponential style functionality might be to define generic form designer functionality meta-data in the IMDME editor for the next version of the IMDME editor that could then be used to define any of the potentially scores of required new forms within future versions of the IMDME editor, e.g. new forms to readily define security privileges, object properties etc.

Note that for the above example of a form designer this could be defined entirely within the current scope of meta-data EIS application functionality as a GUI but non

form-based functionality, or with the addition of a new interactive GUI form drawing object combined with supporting meta-data and interfaces.

One simple example of how the iterative or evolutionary approach, particularly defining exponential style new functionality, might proceed is listed in the following diagrams.

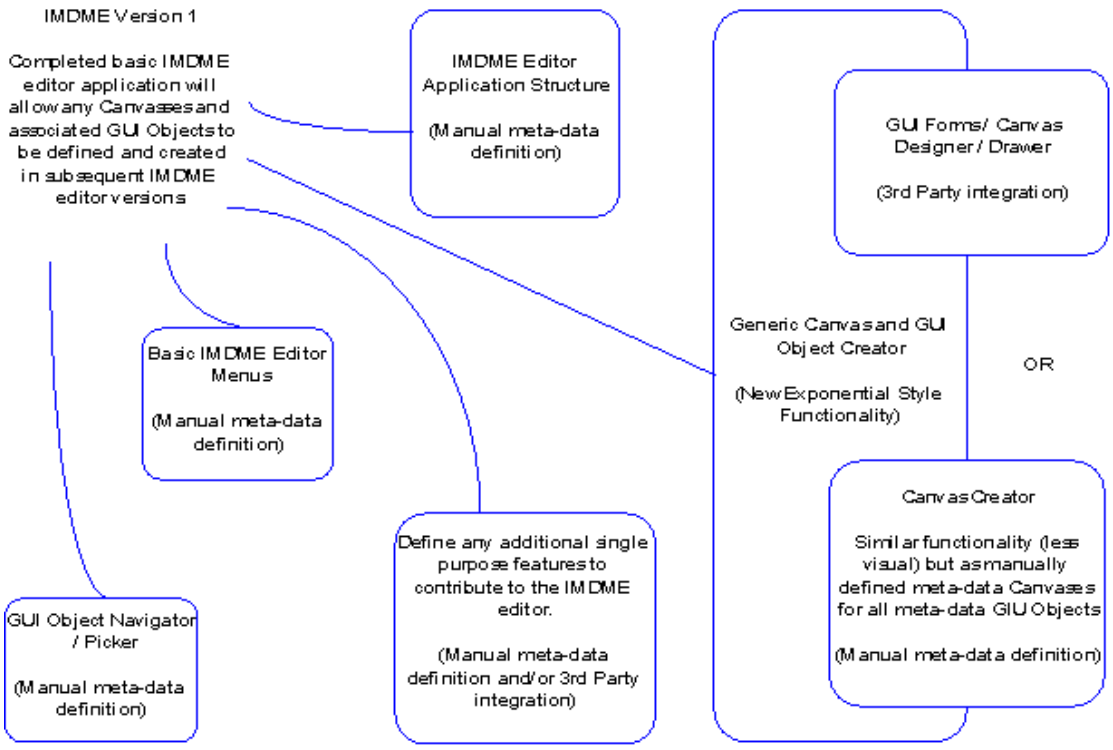


Figure 49 - Initial IMDME Editor Version

Figure 49 depicts a potential initial version of the IMDME editor where the basic meta-data structure and navigation menus and manually defined meta-data. A UI Object Navigator / Picker would be defined as manual meta-data to provide access to objects. The key component, a Generic Canvas and UI Object Creator, as an early example offering exponential style new functionality, would be defined to then allow all Canvases and associated UI Objects to be easily defined through the IMDME editor. As with all complex objects, the option is to define the component through the current functionality of the IMDME editor or to integrate a third party object into the IMDME editor.

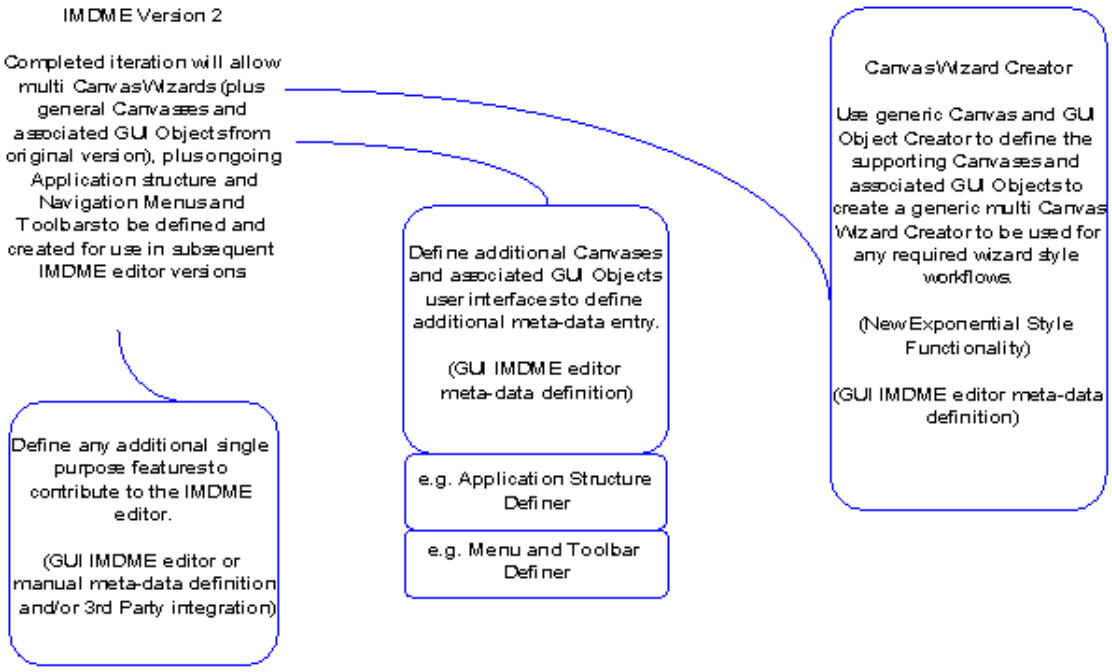


Figure 50 - Second IMDME Editor Iteration

Figure 50 depicts a potential second iteration of the IMDME editor where previous functionality that had been defined as manual meta-data (Application Structure and Menus) has now been provided as standard IMDME functionality by use of the new Canvas and UI Object Creator. This same Creator has also been used to define another example offering exponential style new functionality, the Canvas Wizard Creator, that will define multi Canvas workflows as required. Any other useful functionality can also be added using the increasing IMDME editor functionality.

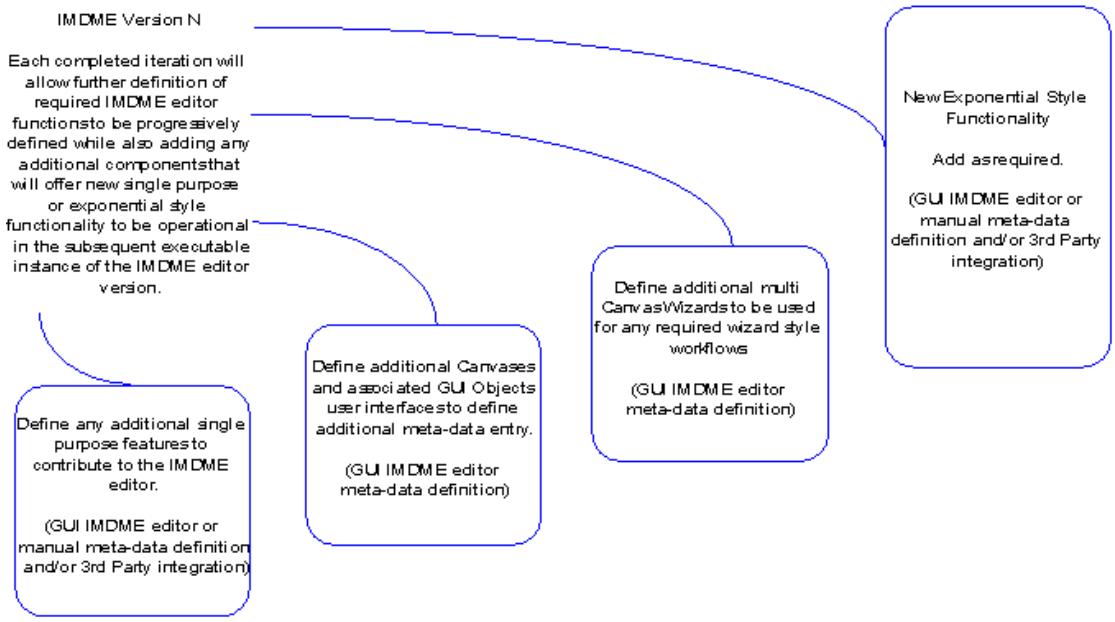


Figure 51 - Progressive IMDME Editor Iterations

Figure 51 illustrates a generic ongoing workflow for each progressive iteration of the IMDME editor, using the existing editor functionality to continue to add new features whilst also adding new single purpose features or exponential style functionality using any combination of objects whether defined by the GUI IMDME editor, by manual meta-data definition or third party integration combinations until all required IMDME functionality is provided.

It would be possible for the IMDME editor to be setup to operate directly on its own meta-data EIS application model to give immediate feedback to the application definer although this may not always be the best recommendation. Some meta-data editing would at times be required to be performed on the meta-data defining the same application functionality that was being used to perform the edit which could result in unknown but potentially loss of functionality consequences.

Although the same temporal meta-data functionality would allow the application definer to wind back to the previous period of satisfactory operation, there may be practical meta-data editing sequences that best require an offline style editing solution. The safest recommendation is to operate the IMDME editor in an offline style whilst engaging in an iterative editing of the IMDME editor meta-data functionality.

This restriction does not need to be imposed whilst editing other standard meta-data EIS applications, which can always be modified whilst live, although practical operational standards might prudently place limits on the extents of any large scale live changes.

7.2.5 Variable Knowledge User Definition Metaphor

One of the fundamental premises of this thesis is that meta-data EIS applications can be created and defined by non-technical users. The majority of the back end work of meta-data EIS applications resides in the establishment of the meta-data EIS application framework, the associated runtime execution engine and the IMDME editor components, rather than requiring a suite of highly skilled computer programmers, integrators and testers.

Rather, business analysts, power users and even normal application users will be empowered via the IMDME editor and the encapsulated meta-data EIS application model and associated business rules to tailor the model themselves.

Most software in usage is designed with a specific end user knowledge set in mind; modern IDEs are targeted at technical programmers, spreadsheets are provided for power users and general usage, word processors are available for common use. The fixed usage metaphor programmed into the software requires the end user to attain a suitable working knowledge as defined by the original software designers. This design requirement naturally has the effect of limiting the access to and usage of most software to the relevant specialists.

The meta-data EIS application framework and any instance of an IMDME editor developed as a meta-data EIS application directly support the provision of alternate application logic that can be readily defined to match a users' level of domain knowledge and technical expertise.

The application of Variant Logic (see 4.8.2 Application Adaptability) can allow different parallel instances of combinations of user interface, business logic and database objects to be defined, whether as minor or major components, to offer application alternatives.

The scope of Variant Logic covers all aspects of the meta-data application model so the alternatives can range widely in terms of complexity and functionality. Some practical examples of applying Variant Logic to a meta-data EIS IMDME editor application are:

- **Terminology**: simple examples such as how we address relevant terms throughout the application. E.g. advanced modellers may prefer a term for an object such as schema while less advanced users may prefer to specify the object as a database or table or as data or as information or as a file etc.
- **Object Availability**: not all objects may need to be used, defined or displayed for all types of users. Removing advanced, unusual, rarely used or confusing objects from a form or canvas may be preferable for simplifying usage. E.g. simplify a form by removing an object that provides the option of indicating if an entry is mandatory, and modifying the logic to default to mandatory to account for the removed option.
- **Merging Objects**: similar to above but rather than removal, replacing the functionality of multiple objects with a simpler set of objects with probably reduced functionality but simpler usage.
- **Object Display**: relocating, re-ordering, emphasising or de-emphasising an object can localise an object according to a user's personal usage of the object.
- **Object GUI Type**: the required information for an object can utilise many different entry or selection formats. Changing object types to a preferred, simpler, or reduced option object may provide simpler operation. E.g. replacing the entry of free-form text with a fixed selection of common options to simplify usage and limit variance.
- **Object Function**: not all objects require the full range of input, definition or specification that is normally possible for an object. Modifying the range, options or calculations of an object can reduce its complexity and minimise the incidence of errors. E.g. the true range of an object may be almost infinite but specifying a more definite range to suit a user can promote more accurate data entry.
- **User Assistance**: additional on screen explanations, guidance or instructions can be added, as well as adding or changing the online help and documentation for users.

- **Operations Workflows**: common application usage patterns can be alternately captured as workflows sequences, wizards or condensed as required to guide users within tighter options boundaries.
- **Visual Appearances**: where objects are used to manipulate or display macro data sets it may be prudent to consider alternate and/or simpler overall visual metaphors. E.g. the use of an Entity-Relationship style schema modeller may be considered incomprehensible for many users whereas simply offering the choice of defining standalone tables or master-detail schemas may be more useable to common users.
- **Advanced Modules**: prevent or limit access to advanced modules or features from users that cannot effectively utilise the features.

As potentially multiple Logic Variants may exist for group or personal usage, as application users gain knowledge and experience in the application and application domain they can be readily assigned access to a more sophisticated Logic Variant that satisfies their evolving knowledge and requirements.

An important note is that it is not the pursuit of perfect or correct terminology, structure or metaphors that needs to be the goal, rather appropriateness for users or user groups to encourage their efficient use of the IMDME editor or any other meta-data EIS applications.

7.3 Batch Definition of New Meta-Data Application Logic

The IMDME editor is an essential component of the meta-data EIS application framework as it expedites each individual meta-data model change as the meta-data EIS application is progressively defined.

Similar to the traditional usage of templates and design patterns, there are many occurrences within the overall set of application and meta-data definition tasks where similar or commonly repeating segments of the meta-data EIS application can be sensibly accelerated by the usage of appropriate batch, template or patterned approaches as part of the overall IMDME.

These approaches can effectively and greatly reduce the usage of the IMDME editor to replicate hundreds or thousands of individual meta-data changes by automating the bulk of the meta-data changes and requiring a smaller subset of tweaks to complete the individualising of the meta-data functionality changes.

It is important to understand that the meta-data EIS application framework supports all of the processing requirements to analyse, create, store and action all aspects of these accelerants where the information is stored within the meta-data model or in any associated and accessible data store. As all subsequent user interface, business logic and data structure elements are themselves defined as meta-data, both the analysis tools and wizards, and any output user interface, business logic and data structure objects can also be defined and created for subsequent execution by the runtime engine.

This ability to use the functionality of the meta-data EIS application model to create and modify other aspects of the meta-data EIS application model is a major capability of the overall framework.

7.3.1 Wizards and Workflow Sequences

These generic approaches to repetitively processing similar meta-data functionality but starting from or using modified parameters, inputs or targets can be defined and captured within their own meta-data logic as part of the IMDME, analogous to traditional purpose specific application wizards or workflow sequences.

There is no definition or need for a specific meta-data wizard definition within the meta-data EIS application model. Any wizard is simply a canvas or subsequent collection of canvases sequenced with appropriate logic. The specific purpose of any application logic that we collectively term as a wizard is purely as has been defined within the standard component objects, functions and object workflows as processed by the runtime engine.

As wizards are just an abstract term for a collective of objects they are accessed and executed via the standard object referencing rules. It is important to note that the IMDME editor will need defined functionality to define and create every form of meta-data in the model however many of these instances will not require the additional sophistication, complexity and multiple steps that may be defined in these wizard approaches.

Some of the expected common wizards that would be defined as meta-data logic within the IMDME and offer significant meta-data definition accelerants are further discussed in the following sections.

7.3.2 Visual Appearance Wizard

Meta-data EIS applications are mainly of a visual type although they can be created with no visual elements, only containing logic processing elements. Whilst the meta-data EIS application framework has not been primarily designed as a batch processing environment it necessarily has that capability as part of an overall EIS application environment. The primary benefit of separating non-visual processing from a meta-data EIS application is in a larger multi-application environment where the separate processing may then occurs across the application boundaries.

There are many aspects of an application's appearance and operation that can be unique amongst different organisations, user groups and users yet they may otherwise all be executing identical meta-data logic.

Such look and feel considerations can be customised on the go by the application definers or could be the result of pre-establishing different option sets that could be used to select and set as default setting the desired look and feel of a meta-data EIS application.

This wizard would operate as:

- A set of new IMDME editor specific tables to record the various options and selected defaults for each look and feel aspect of the meta-data EIS application model's UI Objects,
- A new IMDME editor specific Canvas to manage the look and feel sets ,
- A series of IMDME editor specific Canvases as part of the workflow sequence to specify the look and feel for each application area and/or UI Object.

This wizard would often be the initial starting point within the IMDME editor to create a new meta-data EIS application, or could be used as part of the new application commencement process, or even modified to be applied to an existing meta-data EIS application to modify the look and feel of all defined and future UI Objects.

7.3.3 Menu and Navigation Controls Wizard

The use of menu bars and toolbars are a standard option in most applications and also supported by the meta-data EIS application model. The choice of how to group and/or separate the navigation options within each control type is at the behest of the application definer.

Once the navigation controls are defined their general operation becomes primarily transparent within the meta-data EIS application framework as navigation controls can be automatically enabled or disabled for the majority of operations based on the available control flow and dependencies defined by the current meta-data EIS application model in combination with the current state of the objects. E.g. if there are no available source data for a report then the navigation controls to run that report would not be available. Navigation controls can also be explicitly controlled as required.

Analysis of the meta-data EIS application model structure can also identify the underlying navigational structure of the model's objects which can be extracted into a subset structure of navigation shortcuts to be defined in the menus and/or toolbar controls.

This wizard would operate as:

- A set of new IMDME editor specific tables to record options such as default object hierarchy extraction level and default menu and toolbar navigation objects,
- A new IMDME editor specific Canvas to manage the default settings and default navigation objects,
- A new IMDME editor specific function that populates the default menu and toolbar navigation objects and scans the model's object hierarchy to the specified level to extract and merge the required navigation objects.

This wizard would typically be a part of the initialisation of a new meta-data EIS application within the IMDME editor, and would be used throughout the model definition processes to periodically update the navigation controls. Navigation objects can also be manually changed with standard IMDME editor functionality (to be defined as the appropriate control Canvases) and objects can also be flagged to be excluded from any automatic navigation control inclusion.

7.3.4 Data Structure Wizard

The source data structures that underpin each instance of a meta-data EIS application are of course key to the overall application design and structure. It is of even more importance to the meta-data EIS application framework as analysis and reverse engineering of these data structures can provide significant accelerants to the

automated design and definition of supporting meta-data that is the basis of the meta-data EIS application.

The data structure in the meta-data EIS application model is based on two tiers which can then represent and level of functionality and abstraction as required for multiple purposes and interpretations by different user stakeholder groups.

The first tier is represented by the Virtual Table and Virtual Column objects and is a direct mapping to the objects in physical application database. The second tier is where the abstraction in the meta-data EIS application model starts operating; View Columns become separate objects that map to either a Virtual Column or to another View Column and thus can be further abstracted for a new purpose. View Tables then become collections of View Columns, with the underlying relationships of the component View Columns resolving the ultimate relational complexity away from the users and application definers. View Tables can then be further managed by specifying View Filter, View Group and View Sort objects to control their behaviour and offer further decision analysis to other wizards that can then automatically generate meta-data for the structure of the supporting application.

As with all data modelling toolsets some form of additional visual control to facilitate specifying the object relationships between the data definition objects would be a useful addition to the Canvas definition as discussed in 7.2 Integrated Meta-Data Modelling Environment Editor .

This wizard would operate as:

- An IMDME editor specific Canvas to manage the relationships between the Virtual Table objects, from which key association assumptions will be derived from to support multiple wizards' analysis and automated generation functions,
- An IMDME editor specific Canvas to set all properties of each Virtual Table, including such as clarifying relationship keys and identifiers from the constituent Virtual Columns.
- An IMDME editor specific Canvas to create and manage View Columns, their properties and their source derivations,
- An IMDME editor specific Canvas to create and manage Defined Variable Types and associated Defined Variable Type Conversions,

- Other IMDME editor specific objects to create and manage View Tables, and associated View Filter, View Group and View Sort objects as further described in 7.3.6 Automated Data Grid Wizard .

This wizard is more of an interactive functional editor component than strictly a wizard as its operation would be considered to often be more asynchronous in nature, although certain aspects could invoke the more sequential nature of wizards when objects such as View Tables have been defined.

7.3.5 Automated Canvas Wizard

The standard method of defining Canvases and associated UI Objects as the primary user interface will be to utilise the Generic Canvas and UI Object Creator as described in 7.2 Integrated Meta-Data Modelling Environment Editor . This will allow UI Objects to be added and defined one at a time amongst other UI Objects on the Canvas.

A useful wizard to speed up the creation of Canvases, particularly those that are derived from the data in View Tables, would extract all candidate View Columns and auto-populate the Canvas with an initial layout recommendation based on attributes such as; key, mandatory, type and sub-type. An additional accelerant is to analyse the data structure object hierarchy to identify relationships such as master-detail style relationships and then also create the supporting Canvasses and interactions for each relationship. The finer details of each Canvas and associated UI Object can then be individually manipulated using the Generic Canvas and UI Object Creator as required to achieve the final desired result.

Additionally, a mapping of data types to default UI Objects would be maintained to provide the initial UI Object preference for specific data types and any sub-types based on data content analysis.

This wizard would operate as:

- A set of new IMDME editor specific tables to store default UI Object types for specific data types and sub-types, record the generation preferences for each instance of using the wizard, and to maintain the properties for any generated UI Objects to optionally maintain any manual changes since any previous wizard generation,
- A new IMDME editor specific Canvas to manage each generation’s settings,

- A new IMDME editor specific function that scans the View Table's object hierarchy to identify other View Table relationships, and extract and populate the Canvases and associated UI Objects for each identified View Table's component View Columns based on the model wide defaults and any previous generation history.

This wizard would follow on from any manual definition of View Tables, or following instances of reverse engineering View Tables from any accessible external data sources As the wizard also maintains a history of manual changes to each UI Objects associated with View Columns, the wizard can be executed against modified View Tables to incorporate any new View Columns whilst maintaining all manual changes to the Canvasses.

7.3.6 Automated Data Grid Wizard

Data Grid UI Objects are an essential component in providing visual access to multiple data records. Data Grids or similar objects used in many applications and provided by many third party toolset providers share a fairly common set of attributes to assist in the viewing and manipulation of the listed data. The primary differentiators tend to be the complexity of objects that can be displayed, the objects that can be edited within the grid, and the flexibility of the grid display in terms of columns, grouping and sorting.

The operation of the meta-data EIS application Data Grid UI Object is simplified to a degree in that the View Table object has already been defined with and resolves any multi source data relationships. The operation of View Tables is also further modified by assigning View Filter, View Group and View Sort objects.

The simple assignation of properties to a Data Grid would not require a wizard and be adequately served by a standard properties Canvas. However, a comprehensive wizard to guide through each aspect of defining the Data Grid, View Table, View Filter, View Group and View Sort objects would be commonly used.

This wizard would operate as:

- An IMDME editor specific Canvas to set the key properties of the Data Grid object to govern its overall appearances and operation based on pre-set defaults and/or options,
- An IMDME editor specific Canvas to select and manage the View Table – this would be an instance of the standard Canvas required to manage

View Tables, including the standard master-detail style Canvas to manage constituent Assigned View Columns,

- An IMDME editor specific Canvas to select and manage the View Filters – this would be an instance of the standard Canvas required to manage View Filters pre-loaded with the selected View Table, additionally to identify the options for allowing for either subsequent user manipulation or automatic value selection for all View Filter criteria,
- An IMDME editor specific Canvas to select and manage the View Groups – this would be an instance of the standard Canvas required to manage View Groups pre-loaded with the selected View Table,
- An IMDME editor specific Canvas to select and manage the View Sorts – this would be an instance of the standard Canvas required to manage View Sorts pre-loaded with the selected View Table,
- A new IMDME editor specific function that (when selected as a first time option for the View Table) creates the Data Grid and associated View Table objects based on the selected options plus; View Filters based on options such as all / mandatory / entered, View Groups based on the components of any defined key attributes, View Sorts based on View Groups.

This wizard is a fairly simple example which primarily reuses other Canvases and Panels that would already have been defined for individual object management. The primary additional functionality is any optional invoking of a first time use View Table to create simple associated View Filters, View Groups and View Sorts that can then be readily modified as required.

7.3.7 Application Workflow Wizard

Application Workflow is conceptually similar to the operation of these wizards in that it provides a formal structure and sequence for various operations, manipulations and processing to be defined and subsequently executed. The key difference is that a wizard will simply follow a defined sequence of defined activity, whereas the Application Workflow object has a formal authorisation structure throughout the workflow stages that is used to manage the workflow progress and determine and control whether a successful outcome has been achieved, or to route the workflow to

seek the successful outcome. Application Workflows are also triggered based on specific asynchronous activities within the meta-data EIS application environment.

The wizard for each Application Workflow object will commence with defining the key attributes of the overall Application Workflow object including the initial triggering event, the workflow authorisers, and any other Application Workflows to be invoked following successful or failed operation and the associated determination test.

Each subsequent step in the wizard will refer to a specific stage of the Application Workflow, there being any number of defined stages within a potential network of stages for that workflow. Each stage will require definition of the actions to take, the workflow authorisers, how to determine success or failure and any corresponding consequences, and the relative location and order of this Stage amongst other Stages of the Application Workflow.

Some form of additional visual control to facilitate object relationship management for specifying the navigation and placement of each Application Workflow Stage would be a useful addition to the basic Canvas definition as discussed in 7.2 Integrated Meta-Data Modelling Environment Editor .

This wizard would operate as:

- An IMDME editor specific Canvas to set all properties of the Application Workflow including specifying any functions and to prompt for the navigation and management of each Application Workflow Stage,
- An IMDME editor specific Canvas to set all properties of each Application Workflow Stage including specifying any functions, and the relative location and order of this Stage amongst other Stages of the Application Workflow.

This wizard is also a simpler example which primarily reuses other Canvases and Panels that would already have been defined for individual object management of the Application Workflow and its Stages.

7.3.8 Automated Report Wizard

The UI Report object in the meta-data EIS application model is a close relation to the standard Canvas / Panel objects, in fact it is similarly composed of these objects, enhanced with the grouping or banding features represented by separate Panel objects.

Rather than starting from a blank Canvas a wizard will readily populate a UI Report then allowing subsequent editing of the UI Report format to finalise the required appearance and operation.

Properties derived from the attributes of the included View Columns, View Filters, View Groups and View Sorts will enable an accurate first pass to replicate a GUI Report that initially mimics these features. E.g. View Filter criteria and associated View Columns will be displayed in the header, View Groups will match the bandings Panels, View Columns defined in View Sorts will be prominently displayed within the bandings Panels. Additionally, selected numeric data can be summed at each group level.

Further editing of the details of each band or Panel and associated UI Object will use the Generic Canvas and UI Object Creator as required to achieve the final desired result.

This wizard would operate as:

- A set of new IMDME editor specific tables to store default UI Report style preferences for report format types, as associated template Panels that can be used as the starting point for each defined band,
- A set of new IMDME editor specific Canvases to manage the UI Report style preferences and define each band's settings,
- An IMDME editor specific Canvas to set the key properties of the UI Report object to govern its overall appearances and operation based on pre-set defaults and/or options,
- An IMDME editor specific Canvas to select and manage the View Table – this would be an instance of the standard Canvas required to manage View Tables, including the standard master-detail style Canvas to manage constituent Assigned View Columns, additionally to specify an further associated header / footer, value aggregations etc,
- An IMDME editor specific Canvas to select and manage the View Filters – this would be an instance of the standard Canvas required to manage View Filters pre-loaded with the selected View Table, additionally to identify the options for allowing for either subsequent user manipulation or automatic value selection for all View Filter criteria,

- An IMDME editor specific Canvas to select and manage the View Groups – this would be an instance of the standard Canvas required to manage View Groups pre-loaded with the selected View Table, additionally to specify an further associated matching banding criteria such as banding header / footer, value aggregations etc,
- An IMDME editor specific Canvas to select and manage the View Sorts – this would be an instance of the standard Canvas required to manage View Sorts pre-loaded with the selected View Table,
- A new IMDME editor specific function that (when selected as a first time option for the View Table) creates the UI Report and associated View Table objects similarly to that of the Data Grid wizard function.

This wizard would generally be used as the starting point for all UI Reports to establish the initial report structure followed by manual completion although prudent preliminary definition of appropriate UI Report style preferences and accurate specification of View Tables, View Groups, View Sorts and View Filters can result in a high proportion of successful wizard based first time report generations.

7.3.9 Documentation and Help Wizards

Each object in the meta-data EIS application model has provision for defining descriptive and procedural information for the purposes of inclusion in the generation of either output application documentation or online help assistance. How this information is then collated, formatted and combined with any additional template or style information can be applied in a variety of methods.

Object information designated for use as online help will be immediately available when the meta-data EIS application is executed by the runtime engine. Depending on the defined properties the object help information may be directly available as tooltips style help, and by default for on demand help for the current object as part of the overall model structure collation of all objects' help information.

Additional document structure choice and complementary text and other visual aid objects need to be further defined by the application definers to flesh out any further descriptive information that the application definers require for the application documentation or online help contents.

One simple method to provide this additional structure and content in the IMDME editor is to define a documentation assembler as an example of a UI Report.

The structure of the output documentation is determined by the structure of the UI Report whilst the content is populated from the raw data of the meta-data EIS application model itself. Multiple UI Reports may be required to satisfy the full scope of the required documentation.

Another more wizard based alternative is define a series of wizards or Canvases to prompt for and store the additional documentation structure in new IMDME specific tables. Combined with supporting UI Reports to collate the content this may provide for greater ultimate flexibility.

7.3.10 Overall Application Generation Wizard

As each individual wizard or accelerant is defined, it can also contribute to a part of an overall application generation wizard which would progressively be extended to cover all functionality of defining an initial comprehensive meta-data EIS application model.

Whether the wizard is executing from a blank model or, as discussed in the next section, generating from the design concepts embedded in an existing data schema, this overall wizard will guide the application definer through each wizard as required to progressively build on each previously defined or generated aspect of the meta-data model:

- A set of new IMDME editor specific Canvases to manage the workflow between each of the other wizard and accelerants utilised in this overall application generation wizard,
- A set of new IMDME editor specific functions that reviews and selects the candidate source meta-data for each wizard and sequences the appropriate wizard execution for each subset of meta-data, and any further recursive analysis and/or processing.

As the IMDME editor functionality is completed, including all associated wizards and accelerants, this ultimate application generation wizard will then be enabled with the full ability to automatically generate the meta-data EIS applications, with any manual tweaking and improvement during and after the wizard as required.

7.4 Reverse Engineer Existing Data and Structure

A key paradigm of the meta-data EIS application framework is that the majority, if not all, of an application can be derived from the underlying source data structures and such analysis used to generate the meta-data that will define an appropriate meta-data EIS application to effectively execute transactions. The accelerants described in this chapter all contribute their part to this automated generation capability of the meta-data EIS application framework.

Naturally, the extent to how well the source data is defined and structured will have a strong impact on the effective level of analysis and interpretation that can be gained from the structures. An obfuscated database design that uses table and column names that are generally meaningless to most humans (such as NHD56), or without the use of constructs such as primary or foreign keys, or without specifying appropriate column types for the stored data, makes interpretation difficult for the best data administrator.

Whereas, data structures that do employ well defined structures can provide a wealth of interpretation that can be effectively and automatically replicated in the application meta-data to represent suitable data entry screens, update procedures and reports often requiring only cosmetic or minor modifications to satisfy the application designers' intentions.

7.4.1 External Schema Reverse Engineering

The first tier of data representation in the meta-data EIS application model is provided by the Virtual Table and Virtual Column objects and is a direct mapping to the objects in physical application database. The automated analysis and population of this tier is identical to the type of reverse engineering of database schema algorithms that are commonplace throughout most data modelling tools available:

- Create a Virtual Table for each external database table,
- Create Virtual Columns for each constituent column of the external database tables, additionally:
    - Identifying the data type of each column (these will be mapped to meta-data EIS application model types later),
    - Identifying the relationships between columns as directly implied by the database structure using properties such as; primary key,

foreign key, uniqueness etc. Where these formal schema constructs have not been utilised, analysis of the data between similar columns can be readily performed to identify potential relationships for recommendation to the application definer.

The result of this first level analysis is the initial population of the Virtual Table and Virtual Column objects to provide the basic mapping back to the external database tables and to facilitate additional interpretation and content analysis of the schema with a view to automating the generation of the application meta-data..

7.4.2 Manual Schema Specification

Rather than solely reverse engineering an existing database structure, the option always exists to define a new schema for a new meta-data EIS application directly. This has the benefit of working interactively through the IMDME editor which facilitates starting from the data definition, or from the application interaction design, and any other iteration combination.

When reverse engineering from an existing schema the definition limitations on any database schema, no matter how well designed or documented, can almost always be enhanced by manually reviewing the results of the reverse engineering and improving the derived properties as required to ensure the most accurate basis from which the other application generation will proceed from.

Whilst a perfectly designed and documented database structure may be readily reverse engineered, the meta-data EIS application model requires that additional documentary and descriptive information is provided that will aid in the permanent application documentation and available design knowledge. This aspect of manually revising is expected to optionally occur at any and all steps of application generation.

7.4.3 Virtual Schema Analysis

Once the initial available structure of the database tables has been captured into the Virtual Tables and Virtual Columns, additional automated analysis on aspects of the schema can still be used to investigate and identify potential relationships between and usage of the columns, beyond the basic key analysis of most reverse engineering processes.

7.4.3.1 Analysis of Object Naming Conventions

Well defined database structures will tend to use naming standards that are human interpretable, to at least a useful proportion of the original target audience of the database. However, even a term such as DLY_CST which an English speaking business analyst might initially and readily determine to mean “Daily Costs” is not directly interpretable nor confirmable via an automated process.

However, an analysis of the object names may at least provide part of the answer, which when combined with other analyses may provide greater certainties, particularly when used to gain confirmation from the business analysts or application definers throughout the interpretation process.

Sets of common naming candidates and their common abbreviations can be established for various functional domains. E.g. when preparing an analysis set to identifying named objects that may be related to costing data the following set (in English) might be defined as appropriate search candidates; cost, amount, value, charge, price, sale, figure, direct, indirect, operating, profit, expense, loss, disbursement etc. When combined with an online synonym provider these functional sets can be extensively populated with potential candidates.

In addition to identifying any potential expanded terms and thus identifying potential intended uses of data, they can also be used to infer suggested or recommended data types. E.g. if we have identified CST_DLY as likely to contain “Daily Costing” data then a direct inference might be that such a column should be of a numeric data type (or even currency if it existed) and based on further analyses even suggest further specification such as the range of the exponent and mantissa of a decimal number to model the currency.

Potential abbreviations can also be automatically generated for each search candidate based on algorithms that; remove vowels, truncate the term, or from a list of common or previously used abbreviations.

The use of multiple sets of common naming candidates, each representing a particular knowledge domain or aspect, and applying a weight against each analysis can provide part of the supporting evidence. The ultimate determination of how an object’s use, meaning or function may be assessed could be based on either an automatic interpretation using this style of analysis as one of its components and

specifying decision threshold levels, or by providing the analysis results to an application definer to decide.

Sets of common naming candidates could be readily defined to search for and identify any classification of data objects based on their names, and with hierarchical sub-classification, to further refine the classification to any level of refinement.

7.4.3.2 Compound Object Name Analysis

Many objects' names are based on compound terms so the above identification of probably terms needs to be enhanced to firstly analyse the likely separation of terms or abbreviations and then continue with the individual analyses, followed by a collation of the results into the most likely recommendation combinations.

In the previous example DLY_CST might be a simple candidate as it appears to have a delimiter between the terms, as does "DLY CST" however the usage of DLYCST might not be so clear. Again, the use of recursive processing and maintaining a history of useful and common abbreviations, terms and synonyms can aid in the term identification.

7.4.4 Data Analysis

Additional analyses includes a detailed inspection of each data column to review the efficient selection of data type, identify options for efficient user interface object selection and further refine the potential classification of data.

As an example, the storage of dates as a character type field has often been commonplace in particularly legacy database systems, whereas conversion of such data to a more streamlined, targeted or efficient data type can assist in the ongoing management of, access to and processing of the data.

For the most common data types the following analyses might be performed on the data column to further identify intended usage, processing or storage:

- **Numeric**:
    - **Range Analysis**: can identify whether the data is integer, Boolean or float, identify ranges for data entry validation, identify patterns such as precision exponents (e.g. 2 may indicate currency), recommend a storage format,
    - **Discrete Values**: identify if the data represents a continuum or may use a set of repeated discrete values which can be used to

recommend the style of user interface data entry objects and validations,

- **Date and Time**: analyse whether all data fit within the common date and/or time formatting ranges,
- **Pointers**: sequences or integer data in particular may be alternate record pointers that could otherwise identify or confirm the use of primary or foreign keys, requiring validation against other similar data type columns to aid in relationship identification,
- **Specialist Sets**: particular scientific, engineering, post code, telephone number or otherwise specialist ranges of numeric data may be able to be identified, each requiring a specific analysis test to be performed,

- **Logical**: is generally already a specific enough classification although may require confirmation of the actual storage used for the 0 and 1,
- **Date**: should generally already be specific enough although some range analysis may be required to correctly identify each DD, MM, Y? (year) component, also to ultimately confirm any offset calculation to the year component where legacy pre-2000 or even alternate data ranges have been used (this may well require manual confirmation),
- **Character**: can be used in worst case scenarios as a general catch-all for almost any kind of structured or unstructured data such as:
  - **Other Data Types**: Scans are initially required to determine whether the character data has been used instead of more specific data types such as numeric, logical, date etc. Once consistently identified as replaceable by a specific type, the type specific analysis should then be conducted on the data for further identification and classification,
  - **Long Text**: lengthy character sets consisting of long word sequences may be various kinds of descriptive, comment type text,
  - **Names**: generally consisting of mainly two or three word sets, readily identified by comparison to common language name sets, requiring identification of the internal ordering of the first name, last name etc, and potential separation,

    - **Addresses**: as represented by common addressing template patterns, and verifiable by structure and reference to readily available geographical database content, again for likely separation,
    - **URLs**: any kind of reference, file location identifier,
    - **Multiple Structured Record**: a more comprehensive analysis required to identify patterns that can indicate a fixed or variable structure will require more extensive recursive processing,
- **Binary**: often used similarly to the character type:
    - **(Un)Structured**: can often represent multiple kinds of structured or unstructured data as per the character type,
    - **Specific BLOB**: such as video, audio or any other multimedia or file type, often identifiable by the known header structures of common storage formats.

Whilst the above examples refer to the most basic of data types, further additional rules to identify and refine sub-types such as Currency as a sub-type of Numeric would be readily definable and added to the suite of analysis tools to be defined and performed for each column.

7.4.5 Schema Analysis Object Recommendation Automation

Each aspect of the reverse engineering, schema and data analyses progressively add further accelerants to the automatic generation of the application meta-data, on the macro and micro level of detail. These meta-data EIS application design concepts can be interpreted for every aspect of the application model. Whilst never guaranteed to be perfectly derived, they can provide a first cut application design output quality in proportion to the considered design quality of the underlying data structure.

- **Application and Navigation**:
    - **Application Structure**: would continue derivation upon a base selected application template with initial generation options and defaults,
    - **Navigation Controls**: menu or toolbar items created for each defined Canvas and UI Report, plus any other application defaults that may be specified,
- **Canvases and Panels**:

    - **Canvases**: created for each defined View Table, subsets of UI Objects as UI Tabs that will reasonably place objects based on the available screen displays,
    - **Panels**: created for each set of View Column identifier objects, for the group of View Column foreign key lookup objects, for groups of Assigned View Columns that have been identified as potentially associated due to naming and/or object type, all remaining View Columns, auto-generated Canvas controls such as buttons to execute user commands
- **UI Objects**:
    - **UI Object Type**: derived from; the analysis of and expansion of data object names, Object Type of the View Column, identification of discrete vs random values, range analysis,
    - **Verification Functions**: based on identification of discrete vs random values, range analysis, mandatory based on historical data existence test
    - **Alignments**: automatically set between each level of object once placement is determined,
    - **UI Data Grids**: default display option for each View Table,
    - **UI Reports**: created for each View Table based on View Sorts and View Groups, with auto selection Canvas based on identified primary keys,
- **Data Model**:
    - **Object Type**: derived from the analysis of and expansion of data object names,
    - **View Tables**: created for each Virtual Table,
    - **View Columns**: created for each Virtual Column,
    - **Assigned View Columns**: by default to include each Virtual Column,
    - **View Sorts**: generated for each identified database index, sets of primary keys ordered by most uniqueness,
    - **View Groups**: to match any generated View Sorts,
- **Other / Generic**:

- **Object Titles / Help / Documentation**: derived from the analysis of and expansion of data object names.

These accelerants can contribute to a high degree of application generation automation based on the underlying data structures to create a meta-data EIS application suitable for that set of data structures.

Where multiple meta-data EIS application models have been defined for discrete data structures and purposes the following section aids with how the meta-data models can be readily merged to create larger integrated meta-data EIS applications without the need to recode and port the relevant application source code as is usually required in traditional application development.

7.5 Meta-Data Model Merging

We have seen in the previous sections how the meta-data EIS application model can be created and edited, via manual and automated generation options. Whilst the outputs of these processes alone are expected to produce the expected lifecycle savings there are further benefits to be achieved from the meta-data EIS application framework in the merging of multiple meta-data models to become a larger meta-data EIS application with the combined functionality of the separate meta-data applications.

In a traditional development environment, the solution to merging the functionality of multiple applications would generally result in an assessment of the relative business cases (cost vs benefits) of developing real-time integrations between the applications vs re-developing the source code of (often) the least complex of the applications to merge with the major application. The more disparate the source code of each application, generally the more difficulty, expense or effort is required to achieve the final desired result. In many circumstances, opting for an integration approach (if it is the cheaper option) may only be delaying an ultimate redevelopment option so a careful and accurate lifecycle plan and assessment needs to be performed.

For multiple meta-data EIS application models, application merging is greatly simplified compared to traditional development, as there are no disparate technologies involved, only potentially disparate functionality or logic that needs to be combined to become a single meta-data model and application. I identify the following levels of model merging that offer progressively closer levels of integration between

application models that may semantically overlap even though using discrete logical models:

- **Standard Object Referencing (SOR)**: the simplest meta-data merge option involves creating new references in one meta-data model to existing objects in a second meta-data model to provide access to the application features of the second meta-data model to users of the first meta-data model,
- **Virtual Data Object Mapping (VDOM)**: provides deeper level model merging and integration of similar meta-data objects between multiple models that effectively achieves a rationalisation of the underlying relational data structures,
- **Object Envelopment (OE)**: allows defining an object from one meta-data model as a virtual instantiation of a similar object from the other meta-data model, effectively replacing an object and reducing potential duplication between meta-data models.

By applying any combination of these model merging techniques, high degrees of perceived (to the user) and direct (internal to the meta-data model) integration can be achieved with the resulting combined meta-data EIS application presented as a single application.

7.5.1 Standard Object Referencing

When considering the functionality of the meta-data EIS applications represented by the meta-data models, the simplest and therefore most likely initial model integration points would be expected to be:

- **Merging Navigation Objects**: the menus and toolbars, to create the appearance of a single integrated application. The creation of a new and logically combined meta-data EIS application menu or toolbar access structure can immediately present the impression to the users of a fully integrated suite of applications. Also, the omission of unwanted, inapplicable or duplicated application fragments from the merged meta-data structure is also an important initial consideration when merging meta-data EIS application models.
- **Inter-Model References**: provide simple access to features of the other meta-data EIS application by creating new references or shortcuts to

major functions of the secondary meta-data EIS application model from existing objects of the primary meta-data EIS application model. e.g. defining new interface items such as buttons etc on existing forms of the primary meta-data EIS application to invoke events or initiate other forms using functionality from the secondary meta-data EIS application can provide an even higher level of apparent integration in the merged meta-data EIS application model without yet modifying the underlying core logic of the modelled applications.

While these activities of the SOR process are not strictly limited to model merging and integration activities, as they are just standard meta-data EIS application model editing features, they do represent the simplest methods to readily achieve basic model integration.

Figure 52 illustrates the initial steps in merging separate meta-data EIS application models into a single new model, creating just the basic inter-model integration between the models using only the SOR inter-model linkages.

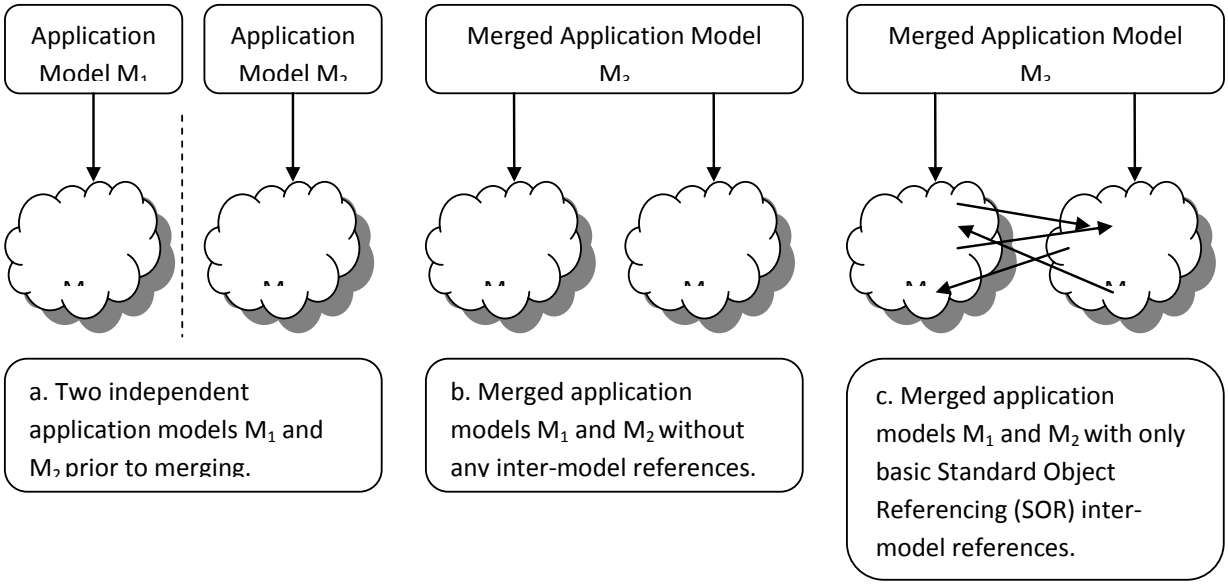


Figure 52 - Standard Object Referencing Model Merging

The management of SOR type model integration options is obviously a fairly simple exercise and is an option that would require the least technical knowledge and

training for a user to exercise. As SOR type modifications do not fundamentally alter the context of the models' individual workflows then they are classed as a safe model modification.

7.5.2 Virtual Data Object Mapping

It is difficult to conceive of two separate applications in the EIS style application domain that have no points of similarity. Indeed the most common entity in any EIS would be a person, project or product, and virtually every EIS application would have some level of object representation of one or more of these in some derivative form. VDOM is the process of identifying similar data elements in multiple meta-data EIS application models, and defining the basic rules that will merge the data elements at the underlying physical level.

While VDOM model integration may have no obvious impact in the apparent execution of the merged meta-data EIS application as perceived by the end-user (which is actually one of its major benefits) it does achieve a progressive integration of the underlying data schema by effectively achieving merging and rationalisation of relationally similar data and thus seamlessly providing access to the combined data for all of the originating applications plus any subsequent model merging or model enhancements.

Additionally the VDOM process then automates the cross application availability of the user interface and workflow support for all associated objects of the newly combined virtual object which may then be observed by application users as new functionality in existing meta-data EIS applications. i.e. the requirement to execute and satisfy each of the objects associated with each of the mapped objects, that initially existed only in the separate meta-data EIS application models would automatically become additional features of each meta-data EIS application which mapped an object, as managed by the model runtime engine.

By progressively identifying and effecting the VDOM process for all similar objects in the merged meta-data EIS application models, a fully integrated underlying database structure, with associated logical workflows and user interfaces is created that services all of the merged meta-data EIS applications without the ongoing inefficiencies that duplication of data causes. e.g. in Figure 53 meta-data EIS application model M1 contains customer information in object JOB while model M2 uses object JOBNUM – by merging these two objects with the VDOM process, the

unioned set of associated objects becomes automatically executable to both meta-data EIS application models including all associated logic and user interfaces required to access, enter and update the associated objects of the entire unioned set.

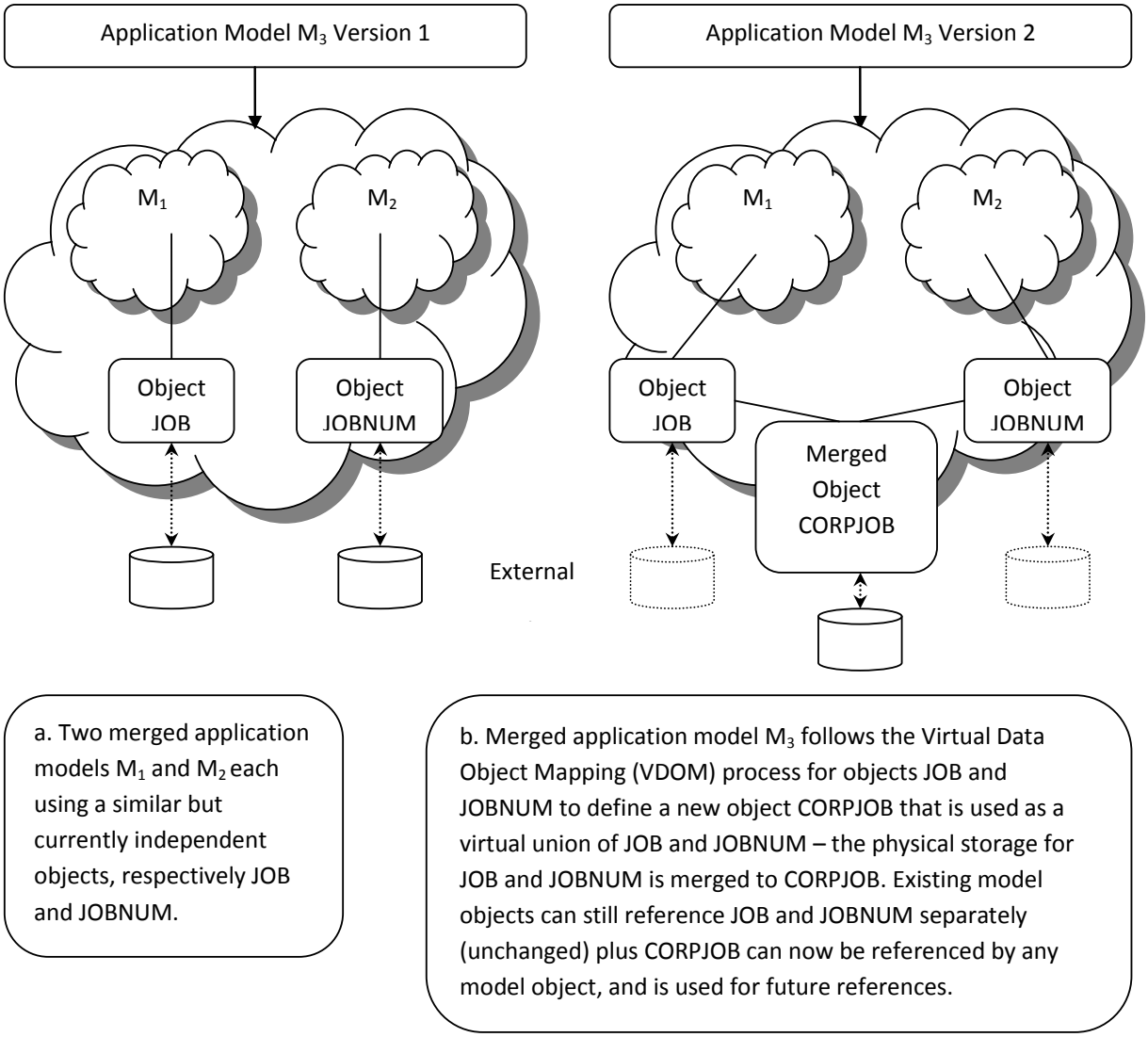


Figure 53 - Virtual Data Object Mapping Model Merging

Following application model merging, data level integration processes utilising the VDOM process facilitates internal data merging at the application model and physical levels, as well as defining the new View Column as a new object to be used in place of the component merged objects for future reference and use.

The VDOM process consists of two steps. The first involves specifying the association functions for the identified objects, which can only be defined as a 1:1 relationship, to ensure valid interoperation of the merged object. A second optional process is required if there is any data in any of the candidate objects, to effectively merge the data and resolve any integrity violations that may be triggered by the new merged object attributes.

- **Step 1 - Merged Object Association**: requires the identification and pairing of objects from each meta-data EIS application model that are very similar - in relational database terms the analogy is creating a 1:1 join between the underlying tables. Each merging object (View Column) must have a direct relationship to only one object (View Column) in the other model – where objects may be semantically composed of multiple virtual sub-objects then new View Columns should be defined to represent the individual sub-objects and facilitate the direct mapping requirement. The original model objects do not require any further specification as all their existing rules are known and remain consistent – the merged object requires only the definition of mapping functions to convert between each of the originating objects. All database constraint rules are preserved and unchanged for the original objects. From the perspective of each original application model, all relationships from the mapped object will automatically execute as required, such as workflows and user interfaces from each application – no changes are required to effect this although user interface aesthetics would be expected to trigger a manual forms edit or merge at some stage rather than rely on the automated generation output from the model runtime engine.
- **Step 2 - Merged Data Resolution**: is only effected when physical data exists in at least one of the mapped objects. i.e. the meta-data EIS application model merging is occurring at runtime in a production system rather than at design time. All existing data in both merged objects is transformed based on the defined conversion functions, including

duplicate detection and any new data migration. The level of automation in this process is dependent on the selection of user intervention options, the level of duplication in the data, and how data may be missing i.e. there may be mandatory View Columns in the second model that were not specified in the other model.

The outcomes of the VDOM process are as follows:

- Each original sub-model is unchanged and still references the original object,
- A new object is created as a merge of the two other objects, which is automatically referenced by the meta-data EIS application model runtime engine, and is available for future referencing within the merged model,
- Any data in the original two objects is merged into the new object. Continuing physical update into the original storage for the original objects becomes optional,
- All associated meta-data EIS application model user interface and workflow objects are automatically invoked for both original objects to ensure all functionality is maintained.

A likely third step in the VDOM process is to "walk the graph" of the merged meta-data EIS application model and merge component user interface objects to provide a more cohesive user interface experience, as well as potentially merge any defined workflows if they invoke some level of apparent user effort duplication. A wizard option could readily be used to automatically redefine all original objects with references to the merged object for completeness resulting in a fully integrated model specification for the benefit of future editing.

The management of VDOM type model integration options requires familiarity with the data structures of the modelled meta-data EIS applications although not strictly at the technical database level. Performing the VDOM process is not analogous to advanced database management as the merged model already contains and abstracts all database details away from the model merging logic definer. Some knowledge of the available system functions is required in order to create the conversion functions as required.

7.5.3 Object Envelopment

When multiple models are merged, in an analogy to the VDOM processes for similar data level elements, other merged model objects may perform similar functions and should also be rationalised into a cohesive merged model structure. e.g. a payroll application model will have numerous instances of a user interface element that is used to select or enter an employee identifier – this application model may be merged with a human resources management application model that would have a similar element in use, although defined completely independently.

OE is the process of identifying functionally similar elements in multiple meta-data EIS application models, and defining the object to be enveloped by another object, usually from an originally different model. The OE process is fairly simple in that it merely requires the identification of each of the objects, and define which object is enveloping (or replacing) the other object. The enveloping object will then be invoked whenever the enveloped object would otherwise have been invoked – requiring a high level of semantic similarity between the objects (see Figure 54 for an example).

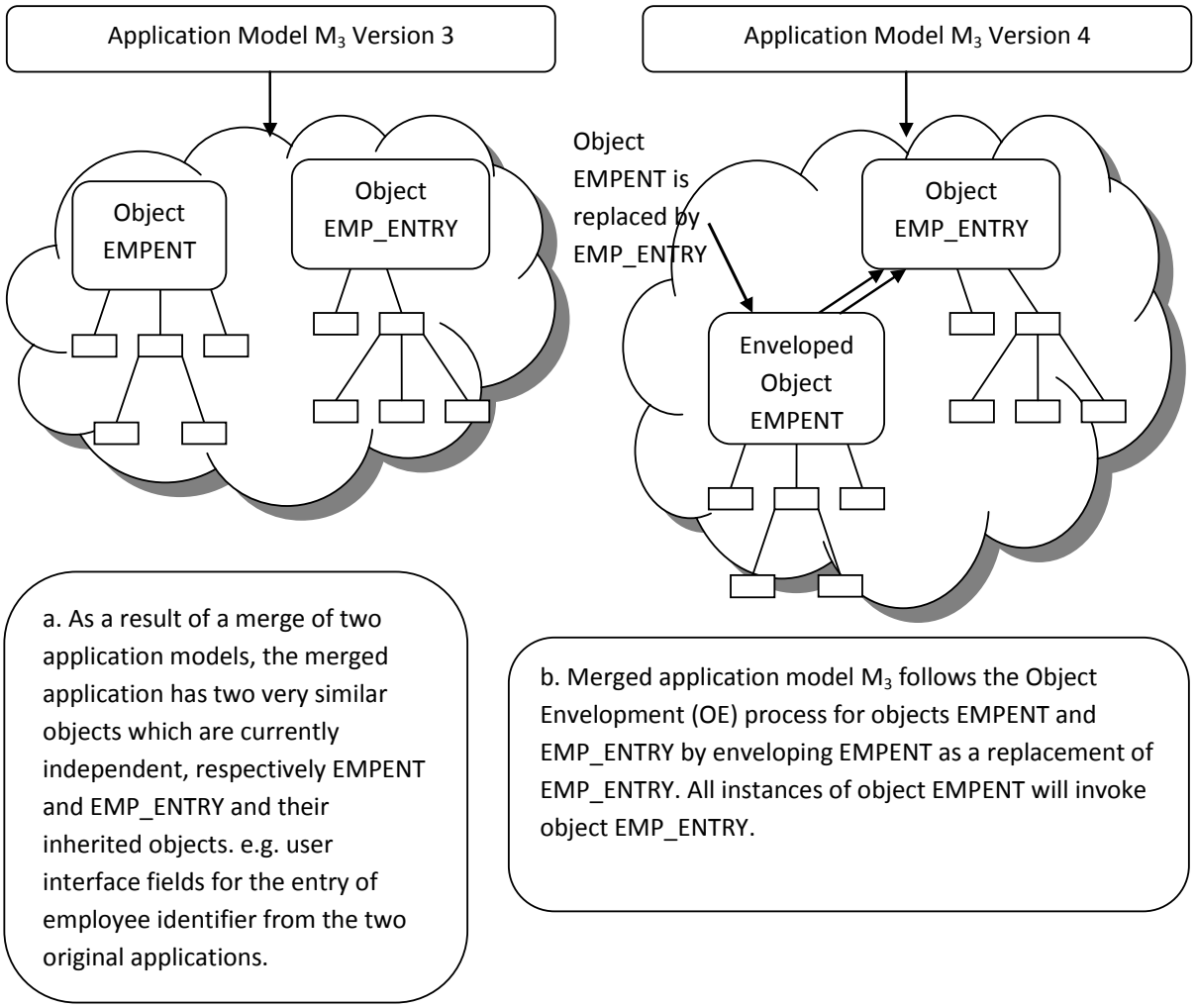


Figure 54 - Object Envelopment Model Merging

The management of OE type model integration requires higher familiarity with the overall model structures of the modelled applications and the defined objects.

7.6 Conclusion

This chapter has described the options available for easily defining the meta-data EIS application models with a greater concentration on how to create and use a GUI based meta-data editor, that I denote as an Integrated Meta-Data Modelling Environment (IMDME).

An interesting and recommended option for creating the IMDME editor is to utilise a recursive development process whereby we initially hand-code a basic

version of the IMDME editor as an example of a meta-data EIS application and then use the first version of this executing meta-data EIS application to then more easily define meta-data and thus further functionality for the next IMDME editor version.

By combining this approach with the creation of purpose meta-data defining wizards (themselves as meta-data model instances), and with any specialist user interface objects to aid in visual data modelling or workflow style tasks, a comprehensive IMDME editor can be progressively created with a potential minimum of effort, whilst itself maintaining all of the lifecycle benefits that any other meta-data EIS application has been demonstrated to obtain.

In addition to the IMDME editor options, which operate on any individual meta-data model, I presented multiple model merging options that simplify how multiple meta-data EIS application models can be readily merged together to provide a single cohesive larger EIS application. The options presented need only manipulate the meta-data model objects rather than requiring any wholesale redevelopment of entire applications or modules as occurs in traditional application development.

In this chapter, I have provided the capability to populate and define the meta-data EIS application models for execution by the runtime engine. The following chapters will complete the logical processing capability by defining the function syntax and web services commands structure and then provide fully detailed examples of meta-data EIS application models to demonstrate the applicability of the supporting framework.

Chapter 8 - Universal Access to Temporal Meta-Data Framework for EIS in the Cloud

8.1 Introduction

The stored meta-data model is the entire basis for the definition and subsequent execution of the meta-data EIS applications. In the preceding chapter we have seen how a logic definer user could use a purpose interactive editor to define the underlying meta-data.

Much of the application logic workflow will rely on the relationships and links between the visual objects defined as the user interface objects – in a wizard based editor environment much of these will be generated automatically based on the underlying data structures. However there will always be the need for additional logical processing to be performed beyond the limited capabilities of induction and deduction of the data schemas.

This chapter concentrates on a command structure that can be used to programmatically communicate direct instructions to the meta-data EIS application

runtime engine and its layers, to both define new meta-data components and to execute meta-data components in response to user or other programmatic methods.

Any IMDME editor (see Chapter 7 - Accelerants for the Iterative Design of EIS Models) will necessarily need to retrieve the meta-data from the model and display it to the logic definer using an appropriate presentation metaphor and design. As logic changes are defined and committed these editor based logic changes need to be translated into the appropriate formal syntax command and submitted to a meta-data EIS application model runtime engine as a structured meta-data model definition command. The runtime engine then processes the logic change, committing if valid or otherwise rejecting.

During user execution of a meta-data EIS application the runtime engine itself communicates between its logical application layers using the formal syntax commands, with the user interface layer capturing the user interactions and structuring the source commands for human users, and a programmatic interface invoking web service calls for remote systems whether instances of MDEIS applications or any technology suite framing the appropriate web service calls and security authorisations. The physical location and combinations of end users and distributed layer components of the runtime engine is immaterial – any widely distributed or cloud based execution is supported by the command structure subject to appropriate internetwork carriage and authorisation between the layer components.

Accordingly, any other programmatic interface can present correctly structured commands to interface to the meta-data EIS application runtime engine from any legacy sources that can; provide the appropriate security credentials, formulate the correct commands, and communicate to the layer component via web services Chapter 6 - Agile Platform for Dynamic Systems Change Management reviewed options for various architecture solutions, this chapter concentrates on the actual programmatic features that need to be available as part of the MDEIS framework.

This chapter details the general Functions that are defined to provide programmatic logic control of the model objects. They are primarily described as Functions and associated syntax as this is the target technology focus to meet business analysts, power users and general application users rather than the highly trained technical specialist software programmers.

Global cloud access to MDEIS instances including individual access to each object’s methods and attributes is then provided by crafting web service calls to

provide secure access to these core Functions from any source, including layer separation of MDEIS runtime or model editor engines, or any external system interface.

8.2 Model Objects Definition and Access Commands

The MDEIS application model objects must first be created in order to be subsequently executed by a runtime engine. This section details the commands available to initially define the model objects and to then subsequently access the model objects from within the desired model logic.

The object definition commands are defined as Functions as they can also be invoked from within the model logic itself as well as from a meta-data editor. Automated logic defining wizards and other reverse engineering utilities or functions would use these model definition commands extensively to progressively create the model based on their interpretations of reverse engineering rules and generated user input definitions.

The object access commands are also Function based as they will be primarily invoked from within user generated logic segments to satisfy the implementation of any identified business logic.

I also include a description of the generic combinatorial syntax rules that apply to all Functions and logical processing statements.

8.2.1 Model Object Creation

There is a common Function and simple generic notation to be followed to create any model objects as follows:

NEW($class_name[“Identifier”]{{.sub_class_name[“Sub_Identifier”]}})

Where:

- NEW() is the system Function to invoke to create the object. It returns the universal GUID identifier of the new object if the object was successfully created or an error code if unsuccessful.
- $ is the system object prefix.

- class_name is the specific model class for which the new object is created as an instance of.
- Identifier is the general textual identifier that is used as a pseudo key for future reference of that object type.
- {{}} denotes optional multiple sub-class components of an object, extended to whatever identification level required for the sub-class.

For instances where an object is composed of multiple sub-classes (which is almost universal in the MDEIS model) the sub-classes are referenced using common dot notation progressively through each sub-class level to be created.

The indicative class names have been represented throughout this thesis in design excerpts of the class and/or ER diagrams. For a full reference to the class names refer to the Appendices which detail the full model extracts.

For example, to define a new Canvas object we would specify a Function like: NEW($Canvas[“Hello World”].

To define a new Tab pane for an existing UI Tab we would specify a Function like: NEW($UI Tab[“Orders”].UI Tab Canvas[“Payment”]).

8.2.2 Model Object Access and Assign

There is a common object notation used to access any model objects as follows:

$class_name[identifier]{{.sub_class_name[sub_identifier]}}

Where:

- $ and class_name and {{}} are as defined in the previous section.
- identifier refers to one of several methods used for identifying the required object:
    - “Identifier” is the general textual identifier that was used as the original defining pseudo key.
    - # representing a numeric row identifier (or simulated array index) for the required object.
    - Row or index manipulation commands such as:
        - START or FIRST to locate the first row or instance of the objects.

        - NEXT {#} to locate the next row or instance in order. If no # is defined the default is 1, otherwise the # modifier is used to skip forward (or backward) by # rows or instances in order.
        - PREVIOUS {#} to locate the previous row or instance in order. If no # is defined the default is 1, otherwise the # modifier is used to skip backward (or forward) by # rows or instances in order.
        - END or LAST to locate the last row or instance of the objects.
    - Unique universal GUID identifier for the object.
    - (function) where function is any logical clause or Function that can be used as search criteria to identify the required object.
- sub-identifier is similar to the above identifier, used for each sub-class access.

For example, to access a Canvas object "Hello World" we would specify a Function like: variable := $Canvas["Hello World"].property where variable is a placeholder for assigning the retrieved value, and property refers to whatever desired property (or sub-class property) of the object that was required.

To access the third Tab pane for a particular UI Tab we would specify a Function like: variable := $UI Tab["Orders"].UI Tab Canvas[3].property.

8.2.3 Model Object Deletion

There is a common Function and simple generic notation to be followed to delete any model objects as follows:

DELETE($class_name["Identifier"]{{.sub_class_name["Sub_Identifier"]}}
{,ALL})

Where:

- ALL is an option to force the deletion of all sub-classes below the identified (sub-)class. If ALL is not specified then a DELETE Function will fail if any sub-classes exist.

For example, to delete a Tab object we would specify a Function like: DELETE($UI Tab[“Orders”]) but this would only work if there were no sub-class UI Tab Canvas objects, otherwise we would need DELETE($UI Tab[“Orders”], ALL).

To delete only a specific Tab pane for an existing UI Tab we would specify a Function like: DELETE($UI Tab[“Orders”].UI Tab Canvas[“Payment”]).

8.2.4 General Syntax Options

As already indicated throughout the thesis, fairly common general syntax rules are utilised for Functions and logical processing:

- **Assignation**: use := to assign a value. Some alternate Functions may provide similar and additional functionality.
- **Arrays**: use array[index] to reference array records where index may be:
    - # representing a numeric row identifier (or simulated array index) for the required object. The default index starts at value 1.
    - Row or index manipulation commands such as:
        - START or FIRST to locate the first row or instance of the objects.
        - NEXT {#} to locate the next row or instance in order. If no # is defined the default is 1, otherwise the # modifier is used to skip forward (or backward) by # rows or instances in order.
        - PREVIOUS {#} to locate the previous row or instance in order. If no # is defined the default is 1, otherwise the # modifier is used to skip backward (or forward) by # rows or instances in order.
        - END or LAST to locate the last row or instance of the objects.
- **Functions**: use nested (,,,) with commas separating arguments.
- **Mathematical Expressions**: use +-*/^ with nested () expressions.
- **Object Notation**: uses nested class_name.sub-class_name to access properties and methods.
- **Object Execution**: the object’s name as a standalone statement passes execution to the object, alternatively specifying an object’s sub-class or a method of the object passes execution to that component.

- **System Functions**: Functions and variables prefixed with $ are pre-defined by the MDEIS framework.

The above syntax rules apply to all Functions and logical statements.

8.3 Pre-Defined Functions and Variables

The MDEIS framework primarily uses a function based syntax to fulfil its objectives of a simplified logical processing metaphor that is more readily accessible to non-technical business users rather than only technical programmers.

This section lists the core Functions that are required to support the model and be supported by the runtime engines and editors.

Some Functions are very specific to the MDEIS model and framework, others are very common Functions that in many cases are in common use in spreadsheet applications such as Microsoft Excel or LibreOffice Calc and may be almost identical in function. Functions that are fundamentally similar or identical in nature to functions used in these spreadsheet applications have been noted with an * in the Function listings.

Where a pre-defined Function does not provide the required functionality a user-defined Function can be defined to provide for any functionality, including accessing external systems via web service calls (see 8.4.2 User Defined Functions).

8.3.1 System Defined Variables

System Defined Variables are automatically set and maintained by the runtime framework engine to simplify the access to some regularly used session and system information. Their use can also increase the readability of user defined functions. They are identified and used as $System_Variable_Name when used in Functions.

System_Variable_Name	Returns
$Application_Id	The main identifier of the current application model in use in the user session.
$Page_Number	The current page number of an executing UI Report.
$User_Id	The main identifier of the currently logged in user.
$Date	The current system date / time.
$Total_Pages	The total number of pages in an executing UI Report.

Table 15 - Common System Defined Variables

Note that users can often achieve similar simplified functionality by creating user defined Functions and invoking then as User_Function().

8.3.2 General Model Processing Functions

These Functions define aspects of the meta-data model application logic. They define or access aspects of the meta-data model.

Name	Arguments	Purpose
ARRAY	Mode, Name, rows, columns, [value, type, persistence]	Define, assign value to or return value from an array of a standard data type, based on the mode (Define, SET, GET), of persistence (session, permanent).
ATTRIBUTE	Mode, Type, Name, attribute, (setvalue)	Returns value from, or assigns value to (as the setvalue), for an attribute of a named and type of meta-data, based on the mode (SET, GET).
DISABLE	(Objects), [condition, persistence]	Disable a set of objects, or optionally based on a condition – the persistence determines whether the condition is executed once or is set as an ongoing disabling test.
ENABLE	(Objects), [condition, persistence]	Enable a set of objects, or optionally based on a condition – the persistence determines whether the condition is executed once or is set as an ongoing enabling test.
LIST	Mode, Name, rows, [value, type, persistence]	Define, assign value to or return value from a list of a standard data type, based on the mode (Define, SET, GET), of persistence (session, permanent).
VARIABLE	Mode, (Names), [value, type, persistence]	Define, assign value to or return value from a standard data type, based on the mode (Define, SET, GET), of persistence (multi, function, session, permanent).
VISUAL	Type, Name, ((Event, Mode,	Calls any visual structural element, of a named and type of meta-data, as though it

Name	Arguments	Purpose
	Function),..)	had been executed, allowing any of events to be executed or over-ridden (Mode is one of; normal, cancel, execute, executenv where the nv suffix means non-visual), and to perform additional remote commands (via Function).
WORKFLOW	Name, View Table, [RecordGUID], [Data Grid], [Canvas], [(Function)]	Calls a defined workflow against a View Table, either all records or against a specific single record as the RecordGUID identifier, and either processing interactively using a defined Data Grid or Canvas, or processing via provided function commands.

Table 16 - List of General Model Processing Functions

8.3.3 Database Management Functions

These Functions define interactions with the underlying application data schema.

Name	Arguments	Purpose
FIELD	Mode, Source, Column, [setvalue], [where]	Returns value from, or assigns value to (as the setvalue), the column, based on where (PREVious row, NEXT row, relative ROW#, based on current sort criteria, default is current), based on the mode (SET, GET).

Name	Arguments	Purpose
LOOKUP*	Source, Column, k	Returns value from the k-th row in a column, based on the current sorting and grouping criteria.
MATCH*	Source, Column, value, k	Returns row of the k-th instance of value in a column.
MOVE*	Source, Condition, [k]	Moves to the (optionally k-th) row in source satisfying the condition.
RECORD	Action, Source	Updates current, inserts a new or deletes current record in the Source, based on Action (UPDATE, CREATE, DELETE).
RECORDDEL	Source, condition, [auth]	Delete records in a table, based on condition (e.g. TRUE() = all, FALSE() = none, other condition to evaluate per record), with specific authorisation (NONE, PROMPT) although may be over-ridden by security and system settings.
SETSORT	Source, sort	Set the current sort for a source based on sort (Named sort, Default, previous).
SKIP	Source, [where]	Changes source record pointer to where (PREVious row, NEXT row (default), relative ROW#, START, END, based on current sort criteria).
SORT	Source, Name, (sort conditions), (group conditions), persistence	Define a named sort and grouping order for a source, of persistence (session, permanent).
TABLE	Name, (fields), (constraints), persistence	Define a table, of persistence (session, permanent).
TABLECLONE	Source, Name, persistence, condition	Clone an existing table to a new table, of persistence (session, permanent), copying records as per condition (e.g. TRUE() = all, FALSE() = none, other condition to

Name	Arguments	Purpose
		evaluate per record).

Table 17 - List of Database Management Functions

8.3.4 Logical Processing Functions

These Functions define statement workflow and logical and conditional processing.

Name	Arguments	Purpose
AND*	Conditions	Returns TRUE if all conditions are TRUE, otherwise FALSE.
CASE	Index, conditions	Chooses a condition to execute based on the index.
FALSE*	null	Returns logical value of FALSE.
IF*	Condition, true, false	Returns the result of executing the true statement if the condition is true, otherwise returns the result of executing the false statement if the condition is FALSE.
LOOP	Test, Condition, function	Loop through execution of the function based on the result of the Condition, the result of which can be tested at either the Start or End of each loop.
MULTI	Functions	Allows any number of functions to be executed sequentially. Can also be invoked as

Name	Arguments	Purpose
		().
NOT*	Condition	Returns the logical opposite to the result of executing the condition, i.e. if executes as FALSE then return TRUE, if executes as TRUE then return FALSE.
OR*	Conditions	Returns TRUE if any conditions are TRUE, otherwise FALSE only if all conditions are FALSE.
TRUE*	null	Returns logical value of TRUE.
RETURN	Value, [mode], [force]	Set or update the value to be returned from a multiple statement function, and optionally force the early termination of the execution, the logical termination scope is based on the specified mode (multi, function).

Table 18 - List of Logical Processing Functions

8.3.5 Group Data Analysis Functions

These Functions provide grouped data processing on data sets.

Name	Arguments	Purpose

Name	Arguments	Purpose
AVERAGE*	Source, Column, [condition]	Returns average of all values in a column, that satisfy an optional condition.
COUNT*	Source, Column, [condition]	Returns count of rows, that satisfy an optional condition.
LARGE*	Source, Column, k, [condition]	Returns k-th largest value in a column, that satisfy an optional condition.
MAX*	Source, Column, [condition]	Returns largest value in a column, that satisfy an optional condition.
MEDIAN*	Source, Column, [condition]	Returns median value in a column, that satisfy an optional condition.
MIN*	Source, Column, [condition]	Returns smallest value in a column, that satisfy an optional condition.
MODE*	Source, Column, k, [condition]	Returns k-th most frequent value in a column, that satisfy an optional condition.
RANK*	Source, Column, number, order, [condition]	Returns rank of number in a column, depending on the order (from HIGH downwards, or LOW upwards) , that satisfy an optional condition.
SMALL*	Source, Column, k, [condition]	Returns k-th smallest value in a column, that satisfy an optional condition.
STDEV*	Source, Column, [condition]	Returns standard deviation of values in a column, that satisfy an optional condition.
SUM*	Source, Column, [condition]	Returns sum of all values in a column, that satisfy an optional condition.
VARIANCE*	Source, Column, [condition]	Returns variance of values in a column, that satisfy an optional condition.

Table 19 - List of Group Data Analysis Functions

8.3.6 Date and Time Functions

These Functions provide date and time conversion and calculations.

Name	Arguments	Purpose
DATE*	Year, month, day	Return internal numeric value representing the date.
DAY*	Number	Return day of month (1-31) of the date represented by the numeric value.
HOUR*	Number	Return hour (0-23) of the date / time represented by the numeric value.
MINUTE*	Number	Return minute (0-59) of the date / time represented by the numeric value.
MONTH*	Number	Return month (1-12) of the date / time represented by the numeric value.
NOW*	null	Return current date / time formatted as per current default date display settings.
SECOND*	Number	Return second (0-59) of the date / time represented by the numeric value.
TIME*	Hour, minute, second	Return internal numeric value representing the time.
WEEKDAY*	Number	Return number (1-7) of the date / time represented by the numeric value, corresponding to the current default start of week settings.
WEEKNUM*	Number	Return number (1-53) of the date / time represented by the numeric value.
YEAR*	Number	Return year (integer) of the date / time represented by the numeric value.

Table 20 - List of Date and Time Functions

8.3.7 Mathematical Functions

These Functions provide specialised mathematical calculations.

Name	Arguments	Purpose
ABS*	Number	Returns absolute value.
ACOS*	Number	Returns arccosine of a number in radians.
ASIN*	Number	Returns arcsine of a number in radians.
ATAN*	Number	Returns arctan of a number in radians.
CEILING*	Number,[significance]	Returns number rounded up to nearest integer, or nearest multiple of significance.
COS*	Number	Returns cosine of a number (in radians).
DEGREES*	Number	Returns number (in radians) converted to degrees.
EXP*	Number	Returns e raised to the power of number.
FLOOR*	Number, [significance]	Returns number rounded down to nearest integer, or nearest multiple of significance.
INT*	Number	Returns number rounded to nearest integer.
LN*	Number	Returns natural logarithm of number.
LOG*	Number	Returns base-10 logarithm of number.
MOD*	Number, divisor	Returns remainder after number is divided by divisor.

Name	Arguments	Purpose
POWER*	Number, power	Returns number raised to a power.
QUOTIENT*	Number, denominator	Returns the integer portion of a division.
RADIANS*	Number	Returns number (in degrees) converted to radians.
RAND*	Number	Returns random number 0<=n<1.
ROUND*	Number, [significance]	Returns number rounded to specified number of digits.
SIGN*	Number	Returns the sign of a number (1 if positive, 0 if zero, -1 if negative)
SIN*	Number	Returns sine of a number (in radians).
SQRT*	Number	Returns square root of a number.
TAN*	Number	Returns tangent of a number (in radians).
TRUNC*	Number, [significance]	Returns number truncated to an integer to the specified number of digits.

Table 21 - List of Mathematical Functions

8.3.8 Character and Text Functions

These Functions provide textual conversion and manipulation

.

Name	Arguments	Purpose
CONCAT*	Text values	Returns multiple text values joined into one text string.
DISPLAY*	Value, mask	Returns text corresponding to the value as modified by the mask (see references for mask types).
EXACT*	Text values	Returns result of test if multiple text strings are identical (TRUE or FALSE).
FIND*	Search, text, start	Returns starting position of the search text within the text starting from a position.
LEFT*	Text, number	Returns the leftmost number of characters from the text.
LEN*	Text	Returns the number of characters in the text.
MID*	Text, start, number	Returns the number of characters starting from the start position of the text.
REPLACE*	Text, start, number, new	Returns the text modified by replacing the number of characters starting from the start position with the new text.
RIGHT*	Text, number	Returns the rightmost number of characters from the text.
REPLACE*	Text, start, number, new	Returns the text modified by replacing the number of characters starting from the start position with the new text.
SUBST*	Text, old, new, number	Returns the text modified by replacing the old text with the new text for up to the specified number of occurrences.
TRIM*	Text, character, where	Returns the text with all leading and/or trailing instances of the character removed depending on where (LEFT, RIGHT, BOTH).
VALUE*	Text, mask	Returns number corresponding to the text as modified by the mask (see references for mask types).

Table 22 - List of Character and Text Functions

8.4 Specialised Functions and Options

Access to all model objects with additional logical manipulation provided by the core list of Functions provide the bulk of specifying the logical processing requirements of an MDEIS application model.

Some additional specifications are required to access advanced functionality that is offered by the overall MDEIS framework.

8.4.1 Defining and Setting the Application Model

The first operation in defining any new application is to first create the Application Model object for that meta-data application model. This would be by a NEW($Application Model[“My Application”]) which also places the session into accepting all future meta-data updates for that new application model until either the session closes or another application model is specified.

To specify an alternate existing application model requires either; commencing the execution of a different application model, via its object as $Application Model[“My Application”], or by invoking its **select** method as $Application Model[“My Application”].select which simply switches execution or editing focus for the session back to that application model with no other explicit execution.

These methods avoid having to include the Application Model identifier explicitly within each meta-data command or update.

8.4.2 User Defined Functions

Any Function code segment can be assigned as a User Defined Function and thus more readily available for global re-use throughout the meta-data model application.

Name	**Arguments**	**Purpose**
FUNCTION	Name, Persistence, ([Common: [arguments]], [Variant: [arguments]]), (body), ([Variant: return function])	Define a function of Name, with Persistence (session, permanent), with Body function commands, and allowing different Variants of (arguments, return function).

Table 23 - User Defined Functions

See 9.3 Preliminary Function Examples for examples of User Defined Functions. The primary case study often uses Functions that could readily be defined as User Defined Functions for more efficient design and reuse.

8.4.3 Variant Logic Functions

The first operation in defining any Variant Logic is to first create the new Logic Variant for that object. This uses the following Function in NEW mode and also places the session into accepting all future meta-data updates for that new version of the Logic Variant until either the

session closes or another Logic Variant is specified. Using the ADD mode continues with updates to the current version of an existing Logic Variant.

To specify an alternate existing Logic Variant requires either terminating the current Logic Variant changes or invoking a new Logic Variant Function (which automatically issues a STOP mode command to the current Logic Variant).

These modes avoid having to include the Logic Variant identifier explicitly within each meta-data command or update.

Name	**Arguments**	**Purpose**
VARIANT	Name, Variant, [NEW, ADD, STOP, END, CANCEL]	Define a Logic Variant for object Name as Variant. Operating Modes are; NEW: defines new version of a Logic Variant, ADD: re-commences changes to an existing Logic Variant, STOP: saves and terminates updates to the Logic Variant for the current session, END: saves and permanently prevents future updates to the Logic Variant, CANCEL: invokes a rollback and cancellation of all updates in the current session and terminates the Logic Variant update.

Table 24 - Variant Logic Functions

8.4.4 Temporal Management Functions

The use of temporal Functions requires that the MDEIS environment is operating in temporal mode. This means that full auditing and tracking of all meta-data and data transactions is occurring, enabling the use of temporal Functions.

Operating in temporal mode is a system level operating setting as it can have major ramifications for the storage architecture and storage requirements. As they are system style commands they also require a form of command interpreter or menu option as part of a runtime environment.

The temporal shifts operate only for the current user's session and reproduce the exact global temporal environment in terms of both the data state and the application state as of the temporally shifted period. Data changes are not permitted when temporally shifted.

Name	**Arguments**	**Purpose**
TIMEBACK	[#]	Invokes a roll back of 1 or # session instructions.
TIMEFORWARD	[#]	Invokes a roll forward of 1 or # session instructions.
TEMPORAL	Date-time | Marker	Invokes a temporal shift to the specified date-time period or to a pre-defined Temporal Timeline Marker.

Table 25 - Temporal Management Functions

8.4.5 Runtime Accelerant Functions

These Functions are provided as runtime accelerants. They provide features that are commonly required as part of the operation of EIS style applications. Accordingly they would be expected to evolve in number and functionality with the maturity of the MDEIS framework.

The runtime execution functionality of these accelerants can be hard coded into the runtime engine or even often readily defined manually as an example of a complex user defined Function.

Name	Arguments	Purpose
LOGIN	[Mode], [Application], [Allow_App_Change], [List_Applications], [User], [Allow_User_Change], [List_Users], [Password], [Attempts], [Attempts_Mode], [Fail_Mode]	Attempt to login to an Application model (default is current). The Mode can be Interactive (default) or Silent although any missing User or Password force Interactive mode. Allow_App_Change is YES/NO and permits the login to allow the Application to be changed, similar for Allow_User_Change (default is NO) and only operate for Interactive login. List_Applications (default NO) allows the application list to be displayed for selection, similar for List_Users. Attempts is the maximum number of permitted login attempts. If exceeded, Attempts_Mode can be HALT to halt all subsequent access, a number (in seconds) to delay subsequent login attempts, or CANCEL (default) to abandon the login. Fail_Mode can be LOGOUT to logout the existing user session upon login failure or MAINTAIN (default) to maintain the original login user session. Returns a logical TRUE if a successful new login session was achieved, otherwise FALSE.
PROMPT	Message_Text, [{Button_Text}]	Display Message_Text and optional buttons displaying Button_Text (default is Yes and No).

Table 26 - Example of Runtime Accelerant Functions

8.4.6 Distributed Execution Request Functions

The Distributed Extension of the MDEIS framework require that inter-instance authorisation requests are managed between the Master and Slave Nodes participating in each Distributed Execution Request.

The definition of each Distributed Execution Request type do not require any specific supporting Functions, other than as may be defined for accelerants, as the definitions are entirely local to the local meta-data model but the invoking of the request to instantiate and manage the operation of the request between the Master and Slave nodes is required once the Distributed Execution Request has been defined.

Name	**Arguments**	**Purpose**
DER	DER_Identifier, ISSUE | SUSPEND | RESUME | CANCEL	Manages a defined Distributed Execution Request. ISSUE instantiates the DER to the Slave node, SUSPEND temporarily suspends the operation of the DER, RESUME resumes a suspended DER, CANCEL permanently terminates the DER.

Table 27 - Distributed Execution Request Functions

8.4.7 Web Service Functions

The MDEIS framework utilise web services to communicate processing requests between application layers or segments of the runtime engines. Authorised web services can thus be used to interact with any object or method of the MDEIS application instance. Logic Definers can also utilise web services to share or request information with other remote applications or invoke remote processing.

Name	Arguments	Purpose
WEB_SERVICE	[Payload]	Invokes a defined remote web service call with an optional over-riding payload.

Table 28 - Web Service Functions

8.4.8 Application Update and Rollback Functions

These Functions invoke an update to the core application model meta-data, analogous to providing an application update. The updates are provided as a consecutive stream of standard meta-data updates.

Updates can be performed on a live system although common precaution would often suggest performing the updates in an off-line state for both the mutual performance of user sessions and the update process, particularly for large and significant updates.

As updates are based on a sequential meta-data stream they can be selectively implemented on a progressive basis.

Rolling back an update can only be performed if the MDEIS instance in operating in temporal mode. Before performing a permanent rollback, the scope of the rollback for MDEIS application should first be verified with temporal rollback to ensure the appropriateness of the rollback. This permanent rollback also affects any data transaction performed after the meta-data updates were performed.

Name	Arguments	Purpose
UPDATE	(Stream), [Version] | [Date-time] | ALL	Accesses updates from a meta-data Stream source. Performs the update to a specific version, to a specified date-time or implements all updates in the stream
ROLLBACK	[Version] | [Date-time]	Reverses or rolls back a meta-data update to a previous version. This can only be

Name	Arguments	Purpose
		performed if operating in temporal mode and its effects become permanent.

Table 29 - Application Update and Rollback Functions

8.4.9 Database Transaction Management

As all underlying relationships between data objects are modelled as the Virtual and View objects, users do not need any explicit data transaction commands or Functions as the runtime engine will execute them automatically in conjunction with the target DBMS whilst managing access to the application data.

Data definition and manual record processing Functions are provided (8.3.3 Database Management Functions), and additional data grouping and analysis Functions are provided (see 8.3.5 Group Data Analysis Functions).

In many simpler cases where UI Data Grids are used to update date, the primary decision will reduce to whether to automatically commit changed data or require specific commit notification.

8.4.10 Security Management

Once established in terms of which users belong to which groups or roles, and the applicable rights and privileges that each role has, there are no further explicit security commands that are required.

Access to the security definition related meta-data objects is via the standard mechanisms already provided, however there would sensibly be security accelerants that would simplify the assignation and definition of the various security relationships. These would logically be a part of both a meta-data editor and runtime environment.

8.5 Conclusion

While much of the overall structure and fundamental data transactions of an application can be deduced and inferred from a well-constructed data schema (see Chapter 7 - Accelerants for the Iterative Design of EIS Models) the finer details and major data processing logic require additional logic to be defined.

In a traditional development environment these details are captured by analysts, designed by architects and coded by programmers. In the meta-data application lifecycle these stages can largely be collapsed into a single stage as the analysts can capture the requirements directly into the meta-data model as both a documentation and simultaneous prototyping platform.

This chapter has described how the MDEIS framework design metaphor is addressed at business analysts and power users, who are often very familiar with both the use of spreadsheets and functions, and the fundamentals of relational data structures, the use of often similar Functions provides almost instant familiarisation with many features. This objective would also reduce the learning curve to allow many normal business application users to progressively tweak the application logic and modify as Variant Logic to provide additional localised optimisations, each with the potential to be made available to other local and distributed users.

By allowing secure access to all Functions and features of the MDEIS model via web services also promotes layer and module separation of the runtime engine plus allowing universal access to and from other remote application and database systems whether executing legacy or MDEIS technologies.

With the ongoing revolution in cloud technologies progressively becoming more fine grained from storage, platforms, servers and applications as cloud services, the ability of the MDEIS framework to effectively support an “Object as a Service” provides another step change in capability.

Chapter 9 - Research Validation – Case Studies for Meta-model Framework

9.1 Introduction

In this chapter I present examples of how the MDEIS framework can be readily used to model applications and subsequently execute them with an appropriate runtime execution engine.

Starting with a trivial "Hello World" example as is done by many application development environment tutorials, I follow with a much larger scale example of a transaction based application system.

The case study analysis has addressed multiple and diverse applications in terms of overall scale, platform and complexity. While it is not practical to document the modelling of an entire EIS application, as the schema descriptions alone can exceed the size of this thesis, I have concentrated on modelling a smaller scale application, the source of which is also readily available and accessible. However, even on this smaller scale, the majority of features of the MDEIS framework will be demonstrated.

Users of Microsoft Access may be familiar with the included Northwind database and application that has been provided as a sample application with Microsoft Access for many versions of the product. Whilst not of EIS class in terms of scale or integrity,

it nevertheless demonstrates many of the necessary EIS features that I wish to demonstrate such as; security, rich user interfaces, logical workflows, transactions and reporting. The Northwind application also has the benefit of accessibility whereas other specific EIS applications are usually relatively closed environments, accessible only to current customers.

Modelling the Northwind application and database captures the most basic essence of the MDEIS framework as I describe the modelling steps in terms of the basic model elements, rather than through the use of accelerants and editors (see Chapter 7 - Accelerants for the Iterative Design of EIS Models) that would have a drastic effect on simplifying the model definition and capture.

The use of Functions throughout a complex EIS will be widespread to capture model logic that cannot be derived from database structure. Many examples of simple and complex Functions are demonstrated in this chapter, both as part of the Northwind application and as further logic segment examples.

9.2 Hello World

Creating a simple application that displays the text “Hello World” has long been a common beginner tutorial for many computer languages and systems. The meta-data EIS application framework can also readily provide this functionality via a few quickly defined objects.

There are 4 objects that need to be defined for this example:

1. Application Model: to define this collection of objects as their own application model “Hello World”.
2. Canvas: to display and organise various Panels.
3. Freeform Panel: to capture the orientation of various UI objects.
4. UI Text: to display the “Hello World” text.

Combined with some basic configuration of each object, this application would quickly achieve its outcome, even when entering the meta-data definition manually without the benefit of an editor.

However, the following larger scale application example will provide a more extensive demonstration of the application modelling capability of the MDEIS framework.

9.3 Preliminary Function Examples

The Northwind application provided many examples of both simple and complex Functions to provide logical processing. Some further examples of complex Functions are provided in this section to demonstrate their use and capability.

9.3.1 User Defined Function: Factorial

The following example is a choice of algorithms that could be defined to compute the Factorial of a given positive integer. i.e. 1 * 2 * ... n-1 * n.

Three options are provided including a recursive solution:

9.3.1.1 As a Standard Inline Function

Assumed to be an inline function with a provided integer variable ‘n’ for which the factorial is required.

IF(n <= 1, 1, MULTI(fact := n, i := n, LOOP(**end**, i <= 2, MULTI(fact := fact * (i-1), i := i - 1)), RETURN(fact)))

Features of this example:

- The use of the MULTI() function to permit multiple sequential functions to be executed, for both the primary conditional execution, and the loop execution.
- The temporary variables ‘fact’ and ‘i’ are temporary to the scope of the MULTI() function.
- The use of a LOOP() to iterate through the successive *(i-1) calculations.
- Specifying a return value to the primary conditional MULTI() via the RETURN() specifying the return value for the local MULTI() which is required. This could also have been specified progressively within the LOOP() although in this example it would have been less efficient.

9.3.1.2 As a User Defined Function

Assumed to be a user defined function with a provided integer variable ‘n’ for which the factorial is required. The core algorithm is very similar to the previous inline example.

FUNCTION(‘FACTORIAL’ , ((n,**int**)) , **int** ,**perm,** IF(n <= 1, RETURN(1, **funct, force**), MULTI(fact := n, i := n, LOOP(**end**, i <= 2, MULTI(fact := fact * (i-1), i := i - 1)), RETURN(fact, **funct, force**))))

Features of this example:

- Define a permanent FACTORIAL function in the meta-data EIS application via the ‘perm’ persistence, so can be used by all system users subject to any specified access permissions.
- The variables ‘fact’ and ‘i’ are local to the function.
- A previously undefined variable can also be used directly without being first specifically defined as it is then typed according to the result of the first expression, however its scope defaults to the current system default which may not always be appropriate for the intended usage.
- Specifying a return value to the primary condition of n <= 1 via the RETURN() has been illustrated although is not necessary in this example as the 1 specified in the previous example would be returned as the IF() value and hence the default returned value. The final use of the RETURN() is used to force termination at the end of loop processing and return the appropriate value from the function.

9.3.1.3 As a Recursive Function

Assumed to be an alternate user defined function using recursive logic with a provided integer variable ‘n’ for which the factorial is required.

FUNCTION(‘FACTORIAL’, ((n,**int**)), **int**, **perm,** IF(n <= 1, RETURN(1, **funct, force**), RETURN(n * FACTORIAL(n-1), **funct, force**)))

Features of this example:

- This particular example demonstrates a much simpler recursive function definition with recursion replacing the LOOP() processing and temporary variables.

9.3.2 Batch Processing of Data: Payroll

The following example implements a batch processing function that creates a regular payroll from a series of timesheets.

9.3.2.1 Existing Timesheet Data Structure

Consider the following conceptual model of a basic Employee entity, which has dependent relationships to; Timesheet which records the individual daily hours worked, and to Payroll which records the collated and calculated payments and totals.

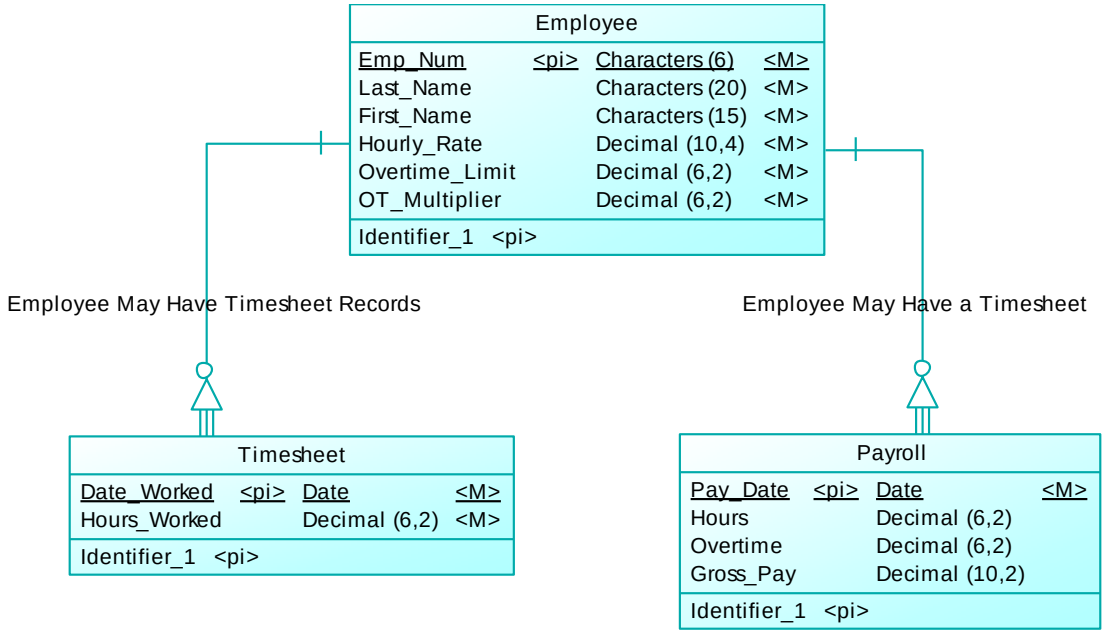


Figure 55 – Example Payroll Data Structure

9.3.2.2 Batch Processing Permanent Function

A permanent function that can process the weekly payroll can be defined as per the following example. This is a quite complex Function that would be expected to be completed by a business analyst, power user or even a programmer.

FUNCTION('WEEKPAYROLL', ((Date_From,**date**), (Date_To,**date**)), logical, **perm,**

MULTI(Emp := '', Normal := 0, Worked := 0, Rate := 0, Limit := 0, Multiplier := 0, Hours := 0, Overtime := 0, Gross := 0, TABLECLONE('Timesheet',

’ThisWeekTimes’, **perm**, AND(Timesheet->Date_Worked <= Date_From, Timesheet->Date_Worked >= Date_To)), TABLECLONE(‘Payroll’, ’ThisWeekPay’, **perm**, FALSE()),

LOOP(**start**, NOT(ISBLANK(ThisWeekTimes->Date_Worked)),

MULTI(Worked := ThisWeekTimes->Hours_Worked, Emp := ThisWeekTimes->Emp_Num, MOVE(‘Employee’, Emp), Rate := Employee->Hourly_Rate, Limit := Employee->Overtime_Limit, Multiplier := Employee->OT_Multiplier,

IF(MOVE(‘ThisWeekPay’, ThisWeekPay->Emp_Num = Emp) = 0, MULTI(RECORD(**create**,‘ThisWeekPay’), ThisWeekPay->Emp_Num := Emp, ThisWeekPay->Pay_Date := Date_To, Hours := 0, Overtime := 0, Gross := 0, MULTI(Hours := ThisWeekPay->Hours, Overtime := ThisWeekPay->Overtime, Gross := ThisWeekPay->Gross_Pay)),

IF(Hours + Worked <= Limit, MULTI(ThisWeekPay->Hours := Hours + Worked, ThisWeekPay->Gross_Pay := Gross + Worked * Rate), IF(Hours = Limit, MULTI(ThisWeekPay->Overtime := Overtime + Worked , ThisWeekPay->Gross_Pay := Gross + Worked * Rate * Multiplier), MULTI(Normal := Limit - Hours, ThisWeekPay->Hours := Hours + Normal, ThisWeekPay->Overtime := Overtime + Worked - Normal, ThisWeekPay->Gross_Pay := Gross + Normal * Rate * Multiplier + (Worked - Normal) * Rate * Multiplier))),

RECORD(**update**, ’ThisWeekPay’), SKIP(’ThisWeekTimes’)))))

Features of this example:

- This is a complex Function and in an editor would be expected to be displayed and edited in a user friendly manner using automatic tabbing and colours etc to aid in logic separation, rather than the plain presentation above.
- Has been created as a permanent function accepting the date range as parameters,
- The use of the first MULTI() function acts to; declare convenient variables, and to use the TABLECLONE() to create temporary working copies of the database as; an extraction of only the required dated timesheet records, and the creation of a blank payroll processing table.
- The use of a LOOP() to iterate through each timesheet record as:

- The subsequent MULTI() commences by extracting the useful data to the working variables,
- The IF()/MOVE() attempts to locate an existing payroll summary record for the timesheet and if that doesn't exist then it creates a new record,
- The final IF()s perform the individual processing for that timesheet record, updating the hours worked, determining if overtime is required to be paid, and calculating the gross payments.
- The final RECORD() updates the changes to the payroll record, and continues through the timesheet records for processing.

9.3.2.3 Invocation and Subsequent Actions

Such a function could be executed from another function or perhaps from a button on the paymaster's screen.

It could also be part of a defined workflow requiring say the printing of a summary payroll report for approval, followed by the printing of the permanent payroll slips for employees and the merging of the payroll records into permanent storage and for costs posting.

9.4 Case Study: A Prototype EIS Management System

There are many types of EIS or ERP systems that are candidates for modelling and simulation as a prototype to verify the potential capability of the temporal meta-model framework. Numerous examples of working application data-based systems were analysed during this research to identify the modelled objects that the framework model currently supports.

Figure 56 below illustrates some of the candidate EIS/ERP application systems that could be readily modelled along with the key functionalities that would be provided by the temporal meta-model framework in the execution of these modelled applications.

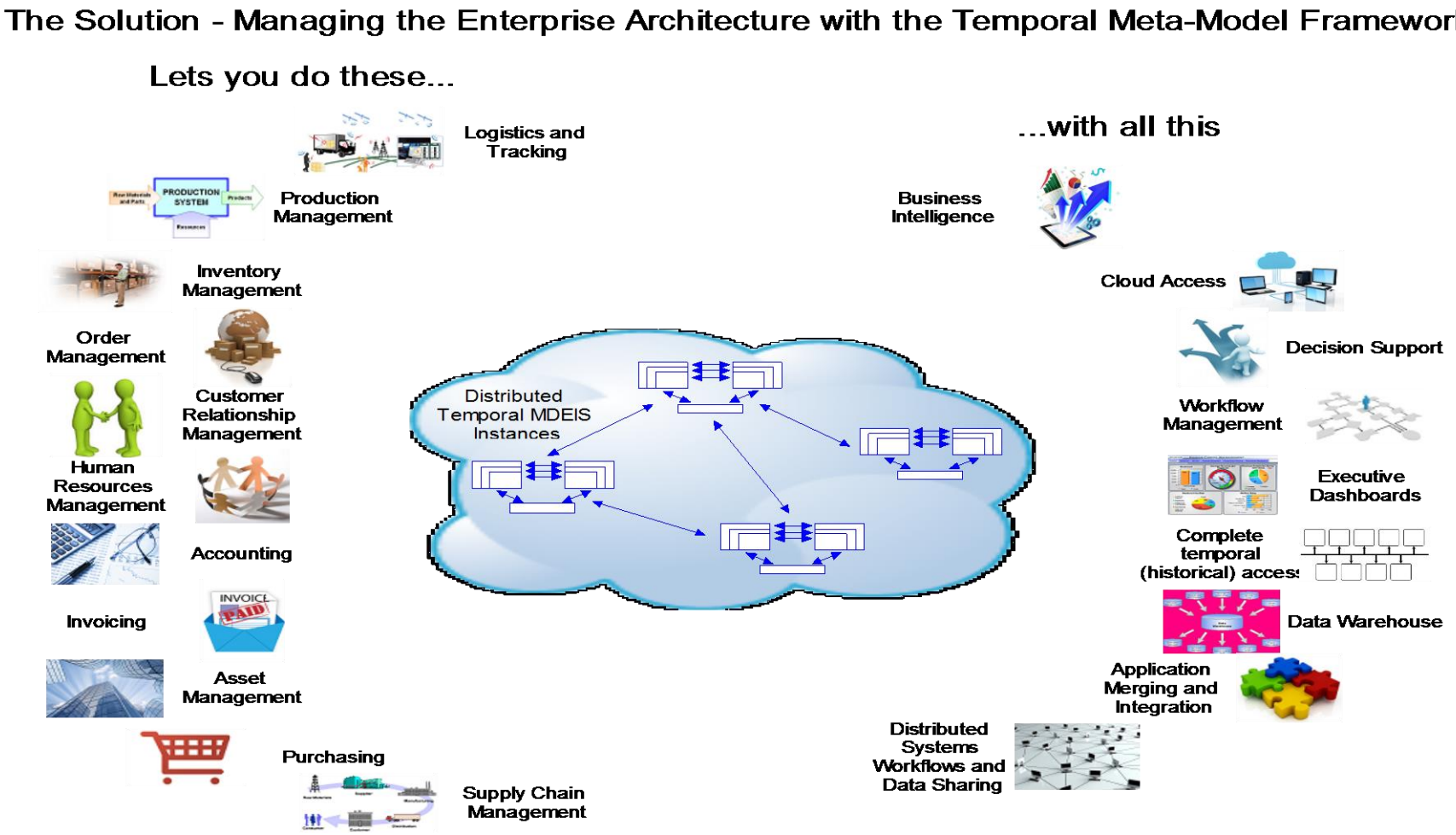


Figure 56 – Candidate EIS/ERP application systems and key framework functionality

While simulating the solutions required for real world EIS applications has been required during the research and definition phase of this thesis, the selection of a suitable, well known and readily understood case study source is provided by Microsoft's Northwind database. A common staple of database systems, the Northwind database has provided as a fully working sample with Microsoft's Access database system for many years to millions of its users. It has also been used in many educational courses and by many other database systems in direct or modified format to also demonstrate their own comparative capability.

In this section I will base the case study on a direct implementation of the current Northwind database as provided by Microsoft's Access 2010. The Northwind database models an order management system and is provided with a supporting application that provides for (see Figure 57 for the main application dashboard);

- Inventory management and purchase from suppliers
- Order and customer management
- Employee management
- Shipper management, plus
- Sales management and reporting.

Our case study will implement similar functionality using the model objects of the meta-data EIS application framework.

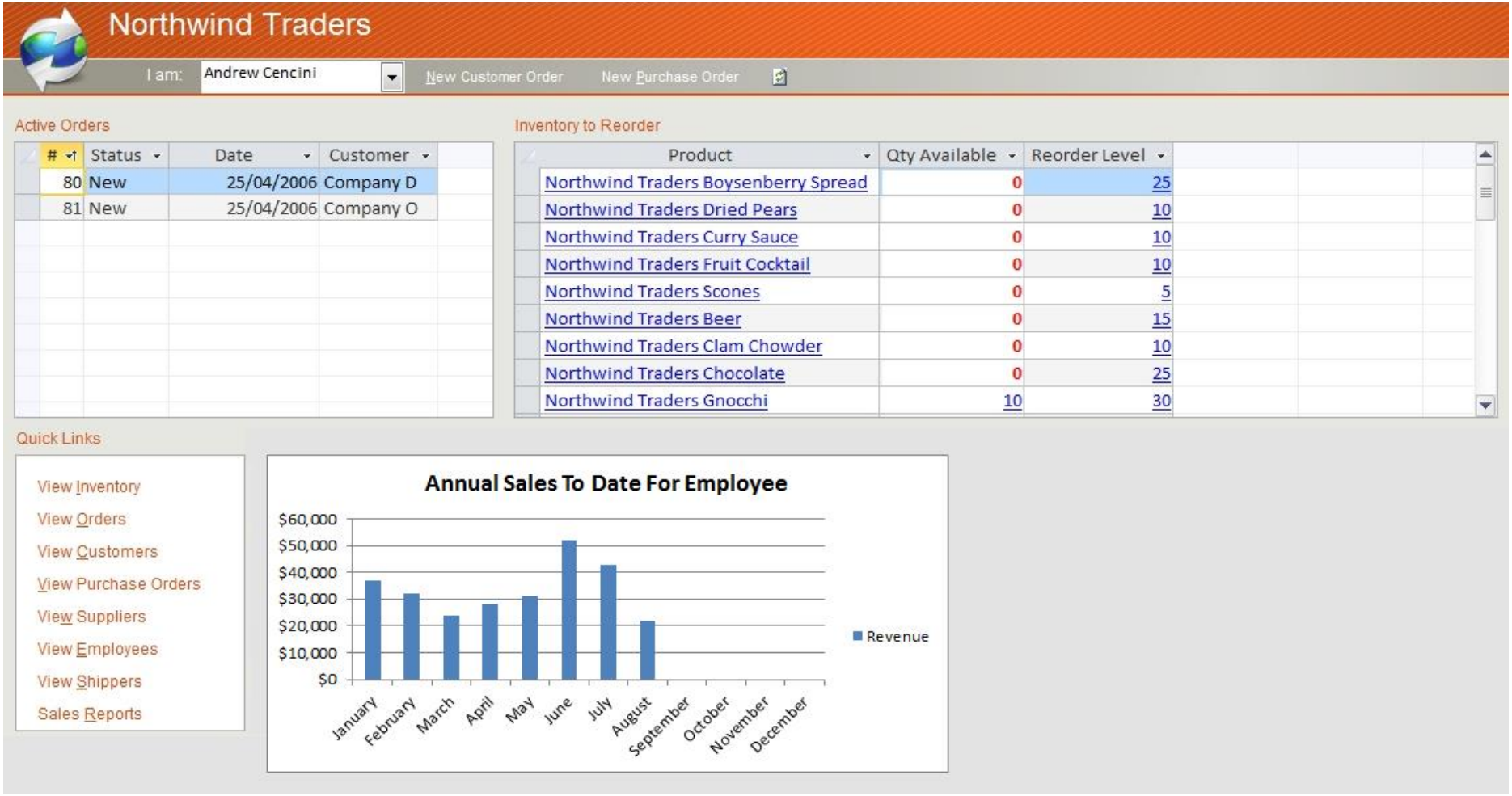


Figure 57 – Northwind Main Application Dashboard

9.4.1 Define Application Model

The first operation is to define the existence of the meta-data EIS application by creating a new entry in the Application Model object. The only entry specifically to define is a suitable COS_Name to name this application, say “Order Management System”, as well as some suitable descriptive text to aid in the generation of the online help and manual documents. For simplicity, I will not generally explicitly state the documentation requirements for each object and assume that if entered they would be limited to the default language.

By default the Display_Login_On_Start would trigger a user login dialog at runtime.

Most of the initialisation and setup tasks would ultimately be performed by the various wizards as detailed in 7.3 Batch Definition of New Meta-Data Application Logic .

9.4.2 Source Data Schema

As the Northwind database represents an existing database schema, this schema would be directly imported into the meta-data EIS application model as per the reverse engineering options detailed in 7.4 Reverse Engineer Existing Data and Structure . The import would ultimately be performed by the runtime engine assuming that it has been supplied with the appropriate database interface connectivity.

For our case study purposes I will highlight the major classes that need to be populated with the target schema meta-data. The primary classes to capture the initial physical schema are:

- Virtual Table and Virtual Column to map the corresponding physical table and columns.
- Virtual_Table_Key and Virtual_Table_Key_Order to map the existence of potentially multiple primary and alternate keys which can also be based on composite keys.
- Virtual_Relationship and Virtual_Relationship_Column to record any foreign key relationships which can also be based on composite keys

Reverse engineering database schema is a well understood field although we have to map the target schema attributes into our structures. I will proceed through

analysing each of the tables in the target schema to record and allocate the schema meta-data.

Note that the listed attribute inheritances in the various meta-data EIS class diagrams will specify many additional attributes that need to be defined automatically for each physical instance of data tables. If the data tables are defined through or managed by the meta-data EIS runtime environment the additional attributes will be defined automatically. Alternatively, separate meta-data EIS data instances will need to be created (and mapped through to the originating data) to allow the meta-data EIS runtime environment to effectively fully manage the data temporally.

Figure 58 below is an extract of the original database schema.

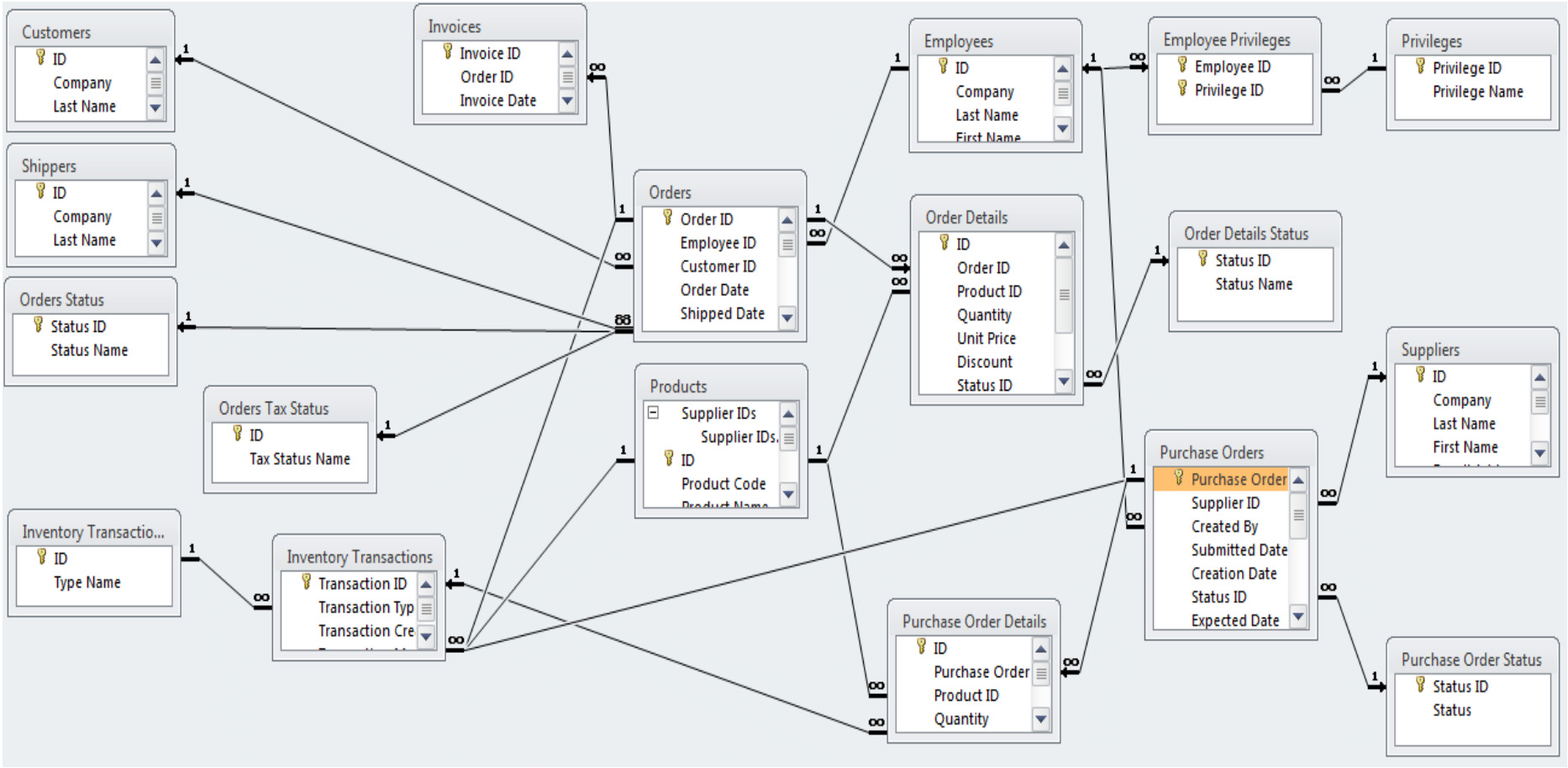


Figure 58 – Northwind Database Schema

9.4.2.1 Simple Tables

Firstly we would consider the simplest of tables, those with no relationships at all, however there are none in this example so I will then proceed to map the next simplest tables – those with no foreign key relationships.

These tables do not require any entries in the Virtual_Relationship and Virtual_Relationship_Column classes. They include:

- Customers
- Shippers
- Inventory Transaction Types
- Orders Status
- Orders Tax Status
- Products – although has an implied 1 to many relationship with Suppliers which will be modelled separately later
- Employees
- Order Details Status
- Privileges
- Suppliers
- Purchase Order Status

For simplicity, as the process is identical for each of these input tables, I will choose one table to highlight the meta-data definition – the Customers table. Initially a new entry for Customers is made in Virtual Table.

Then for each of the listed Customer columns a new entry is made in the Virtual Column table related to the Customers Virtual Table, also setting the Defined_Variable_Type for each (we will assume that all common types are already suitably defined) and specifying the Virtual Column size attributes. Additional attributes to be set are:

- **ID**: is the only listed primary key. A new entry to identify the key is made in Virtual_Table_Key with a Key_Order of 1. The ID Virtual Column is then selected as the sole Virtual_Table_Key_Order entry with a Column_Order of 1. As ID is set to auto-increment then we need to specify an instance of the Field Entry Generation Rules (SEE NEW REFERENCE) defining an incrementing range for this Virtual Column.

- All remaining columns can be defined directly as there are no special attributes except for Attachments which has an implied 1 to many relationship which will be modelled separately later.

All of the above listed source tables will have now been “imported” and defined within the first level “Virtual” classes.

9.4.2.2 Implied Tables

Next we need to interpret the existence of any table structures. Generally this would not necessarily have to occur when importing directly from a physical instance unless we were attempting to further normalise and optimise a schema. However as I am also illustrating derivation and reverse engineering from a conceptual schema there may be instances to consider.

There are two main categories to search for:

- **Internal Relationships**: such as the Attachments form the Customers table (and in Shippers, Products, Employees and Suppliers. Each of these represent the need for a new table with a 1 to many style relationship.
- **Many to Many**: where only the relationship has been defined causally without any explicit requirement for additional columns. These represent a new table with 1 to many style relationships to each identified relationship table. The Supplier IDs listed in the Products table represent an instance of this relationship.

The following Virtual Tables will need to be created as:

- **Attachments**: to be created as a common Virtual Table, with Virtual Columns created to; provide a key identifier, and refer to the stored files attributes and raw storage.
- **Product_Supplier**: new Virtual Table requires no additional Virtual Columns.

We will create the required foreign key identifiers in the next section.

9.4.2.3 Foreign Keys

All remaining key relationships have now been condensed to identified foreign key relationships that each need to be implemented as 1 to many style relationships wherever they have been explicitly defined or implied in the conceptual relationships, or as derived from the physical table definition reverse engineering.

Each 1 to many style relationship will be implemented as instances of the Virtual_Relationship and Virtual_Relationship_Column classes as:

- **Virtual_Relationship**: create an entry listing the specific cardinality of each end of the relationship (0-n, 1-n, 0-1, 1-1, n-n) and specify if there is a specific dependency in the relationship.
- **Virtual_Relationship_Column**: for each Virtual_Relationship there may be multiple instances of Virtual_Relationship_Column where the foreign key is based on a compound key.

The full Virtual_Relationship list required is based on each of the listed foreign keys in the original conceptual diagram (for the Invoices, Orders, Inventory Transactions, Order Details, Purchase Order Details, Employee Privileges, Purchase Orders tables) plus those identified in the previous section based on implied tables for Attachments and the new Product_Supplier.

This completes the initial definition of the "Virtual" classes based on the physical schema reverse engineering.

9.4.3 Model Data Schema

To be usable with in the model requires a mapping from the "Virtual" classes to the "View" style classes and then utilising the specific additional schema classes and attributes to further define useful application features.

9.4.3.1 Initial View Column Mapping

By default it is simplest to initially create a View Column for each Virtual Column. This now makes every physical schema element available within the model. A View Column can be composed of multiple Virtual Columns, or parts of View Columns, based on a Function – a reverse transformation Function is also required to ensure data integrity of the Virtual Column. The mapping of a single of multiple Virtual Column to the View Column is specified as entries in View_to_Virtual_Column_Map. In this example only direct single mappings are required.

Wherever required or desired the View Column can be used to alias or re-define any key attribute as part of the View Column definition – which will then ultimately cause an automated 2-way transformation to be applied between a View Column and its corresponding Virtual Column during execution. Additional View Columns can

also be defined as aliases to other View Columns for further abstraction and transformation as required, all ultimately relating back to the originating Virtual Column.

9.4.3.2 Initial View Table Mapping

The next task is to map Virtual Columns to Virtual Tables. This mapping is not always just a simple replication of a Virtual Table to a View Table as we only need to define View Tables to satisfy specific data entry, reporting or processing requirements. Further, not all Virtual Columns will need to be explicitly listed as corresponding View Columns in View Tables as the relationships between View Columns will be derived from the underlying core Virtual_Relationships that have been defined.

However, given that a fundamental requirement in this example is to maintain all data for all schema tables then it is appropriate for a corresponding View Table to be defined to capture all data required from each Virtual Table. Further, we will extend the basic definition of each initial View Table to incorporate more readable View Columns from their foreign key relationships so that each View Table will readily provide access to all the required data. E.g. instead of including just the Shipper_ID we could include the Shippers Company or other details to make the data more readable.

View Tables are assigned View Columns by mapping them as entries into the Assigned View Column in a preferred View Column ordering.

The initial set of View Tables will be defined as:

- A View Table will be defined corresponding to each Virtual Table to capture the basic structures. Add each View Column from an originating Virtual Table into the Assigned View Column for that View Table. Additionally, wherever there is a foreign key also add the parent tables basic textual identification columns. E.g. if Customer_ID is a foreign key then include the Customers Last_Name, First_Name and perhaps City etc.
- Examine each Virtual Table to identify other useful View Columns that can be assigned to make the View Tables more readable and useful by following foreign key relationships. E.g. Invoices has the Order_ID foreign key which may be better served by including other View Columns such as Order_Date, Shipped_Date, Employees Last_Name and

First_Name, plus the Customer's Last_Name, First_Name. i.e. adding View Columns from anywhere along the foreign key chain.

- Repeat these steps for any implied Virtual Tables that were created as a result of analysing the conceptual design relationships as in 9.4.2.2 Implied Tables . In this example, the Product_Supplier and Attachments Virtual Tables would require View Tables. As we are using a common Attachments Virtual Table that records all of the attachments for access by multiple other Virtual Tables we need to consider some additional options as follows.
- A Virtual Table like Attachments that shares aspects of its structure can be modelled in a variety of ways. One method is to have a single View Table with multiple View Filters, each View Filter providing a row based restriction on the data – the downside of that option in this case is that all of the foreign key View Columns are still accessible in its usage (even though blank in each case). A cleaner method is to create a separate View Table for each separate use case. i.e. create a Shipper_Attachments View Table with the common attachments View Columns but only the Shipper_ID and associated readable foreign key View Columns, and create separate View Tables similarly for each other attachment usage. A View Filter should still be created for each View Table to filter out attachments that are not for the current foreign key type.

Now that View Tables have been created to provide access to all known data, and in more readable formats we will only need to access View Tables and View Columns for the remaining modelling.

9.4.3.3 Improve Data Readability

With all currently known data now accessible via View Tables we need to prepare for any known data presentations and access. View Filters provide a selection criteria to be defined as a Function, analogous to a SQL WHERE clause. View Sorts provide a multi-level sort criteria to be applied based on either Functions or Assigned View Columns at each level. Any number of View Filters and View Sorts can be defined and applied as required at runtime, including defining default objects.

Initially, View Filters are only required when there is a known selection requirement. The abovementioned multiple Attachments View Tables are the only

initial requirements for View Filters. Other requirements will likely be identified during further model definition of the requirements.

View Sorts will be initially required to provide the basic sorting for each View Table. The temptation may be to create View Sorts based on the implied primary key View Column of each View Table but this is unnecessary as the runtime engine will determine the underlying updates and optimisations as required, not to mention the ultimate back-end DBMS. View Sorts are only required to aid in the presentation of data and should be defined accordingly. E.g. for the Customers View Table I may initially define a View Sort based on (Company, Last Name, First Name, City) as a first estimate for presenting this data.

While a View Sort level can be defined as a Function it is likely that the initial set of View Sorts will be based only on View Columns until more is understood about specific user or processing requirements. Recall that a View Column can also be defined on the basis of a Function which as a global feature may have more appeal if relevant.

With the initial set of View Filters and View Sorts created this completes the initial data modelling. As noted in Chapter 7 - Accelerants for the Iterative Design of EIS Models the majority of this modelling so far could easily be performed directly by appropriate reverse engineering and wizards.

9.4.4 Define Application User Interface and Logic

We can now commence with modelling and creation of the user interface objects that will be used to allow the user to interact with the modelled data. I refer to the user interface classes defined in 5.3 Visual Structure Elements .

The Northwind application main page consists of a status dashboard depicting:

- current orders,
- required inventory re-orders,
- quick links (acting as a partial menu),
- a highlight to the current user plus some links to invoke data entry, and
- a graph of revenue.

It is portrayed in Figure 58. Additional data entry screens and reports have been defined and are also available for the major table data.

9.4.4.1 Initial Canvas for Application Dashboard

To replicate similar functionality we will need to define a Canvas and populate it with other user interface objects. A Canvas object “Dashboard” is created which is set to be the first Canvas to be executed by the Application Model object. The displayed title will be set to “Northwind Traders”.

The major visual components of a Canvas are Freeform Panels and Navigation Panels. We will also create the following panel objects:

- Freeform Panel “User”: to display the current user,
- Navigation Panel “New Orders”: to invoke data entry for New Customer and New Purchase orders,
- Freeform Panel “Active Orders”: to list and access current orders,
- Freeform Panel “Inventory to Reorder”: to list the required inventory re-orders,
- Navigation Panel “Quick Links”: to provide the quick links,
- Freeform Panel “Total Revenue”: to display a graph of revenue.

Each Panel will be anchored to the “Dashboard” Canvas and other Panels based on appropriate UI Alignment Rules to provide for the minimum, maximum and resizing options as the objects are manipulated by users.

9.4.4.1.1 Freeform Panel “User”

The “User” Freeform Panel has two objects; a simple UI Text object to display “I am:”, plus a UI Selection object “Select User” to display the list of users and allow selection of a new user – for this object we will also link it to the session user.

Figure 59 – Freeform Panel “User”

To define the UI Selection object “Select User” we will follow these steps:

- Create its Selection Type as a “Drop-Down List” to select one predefined user. For its data source we will define a new View Table “Users” rather than a pre-defined list:

- based on the following View Columns (listed as source View Table->View Column for ready identification only); “Order Number” as Orders->Order ID, “Order Status” as Orders Status->Status Name, “Order Date” as Orders->Order Date, and “Customer” as Customers->Company respectively. Note that we only need to select these View Columns - the internal model logic will resolve the appropriate database queries to manage the data based on their pre-defined relationships.
- based on existing meta-data system View Columns from the View Table corresponding to the “Security_User_Account” which lists all application users. While we could reference these existing system View Columns directly, for clarity we will choose to define new View Columns for our application model; “User ID” as Security_User_Account->Identifier (not visible), “First Name” as Security_User_Account->SUA_Name_First (visible), and “Last Name” as Security_User_Account->SUA_Name_Last (visible). Each of these new View Columns will then be assigned to the View Table “Users” as entries in Assigned View Column. Note that View Columns constitute the definition, Assigned View Columns are the instances occurring in a View Table.
- A View Sort will be defined, in this case sorted on “First Name”.
- The “User ID” View Column will be identified as the returning value “Drop-Down List” Selection Type.

- Two Functions will need to be defined for the UI Selection object:
  - For the Initialisation event: retrieve the current user and set the UI Selection object to this value, using:
    - UI Selection.[Name,“Select User”].run.selected=#Current_User_Id
    - Where:
      - “run.selected” refers to setting the runtime attribute “selected” value for the “Select User” drop-down control, and

- #Current_User_Id is a framework shortcut parameter to identify the main identifier of the current logged in user. This would otherwise be a direct lookup from the runtime tables.
- When set, the “Select User” drop-down control will set its current value to match the provided user identifier and display the associated usernames.

- For the On_Change event: attempt login to the new user remaining logged in as the current user if the new login attempt fails, using:
  - IF(LOGIN(Interactive, #Current_Application_Id, No, No, , Yes, Yes, , 3, Cancel, Maintain), UI Selection.[Name,“Select User”].run.selected=#Current_User_Id)
  - Where:
    - System LOGIN() function is invoked to list the current application users and offer an alternate user login. This is a useful system Function however its functionality can also be created based on an application model segment.
    - If the user login fails then the current user session is maintained, no other actions.
    - Otherwise, the new login session occurs (automatically executing any defined session closure logic) and the “Select User” drop-down control will set its new value to match the new current user.

The “I am:” UI Text object will be anchored to the top left of the Panel, with no resizing options. The UI Selection object “Select User” will be aligned by vertical centre with “I am:” and anchored to its right (plus a few pixels), with some variable resizing options set to allow partial shrinkage and expansion of the object horizontally only. The right bottom of “Select User” will also be anchored to the right bottom of the Panel.

The “User” Freeform Panel will be anchored to the top left of the “Dashboard” Canvas for resizing purposes.

9.4.4.1.2 Navigation Panel “New Orders”

The “New Orders” Navigation Panel has three Navigation Panel Item objects; two text icons “New Customer Order” and “New Purchase Order”, plus an icon to refresh the display of all UI Data Grids on the Canvas.

Figure 60 – Navigation Panel “New Orders”

To define the Navigation Panel we will follow these steps:

- Create “New Customer Order” as a text based Navigation Panel Item.
- Create “New Purchase Order” as a text based Navigation Panel Item.
- Create “Refresh” as an icon based Navigation Panel Item. This requires specifying the icon image file to be used.
- Each Navigation Panel Item will be aligned in left to right order from first to third with the first aligning to the top left of the Panel, and the last aligning to the right bottom of the panel.
- At this point, do not identify the model objects that will be invoked by actioning either the “New Customer Order” or “New Purchase Order” Navigation Panel Items – we will define these later once the forms have been defined. We will now set the click event of the “Refresh” Navigation Panel Item as a simple Function to invoke the Refresh_Data event of the Canvas – this will automatically invoke the refresh of all child controls within the Canvas and its Panels, as “Canvas.[Name, “Dashboard”].run.refresh”.

The top left of the “New Orders” Navigation Panel will be anchored to the top right of the “User” Freeform Panel for resizing purposes.

9.4.4.1.3 Freeform Panel “Active Orders”

The “Active Orders” Freeform Panel has only one object; a UI Data Grid object to list and access the current orders for the currently logged in user. It provides the

following features; the Status of an order cannot be changed within the grid display directly but does invoke the dropdown functionality to view the possible options (this is probably unintended behaviour as it then doesn't allow any changes but we will mimic it as illustrative of functionality), and double-clicking on either the Order Number or Date cells opens up the corresponding Order form for editing.

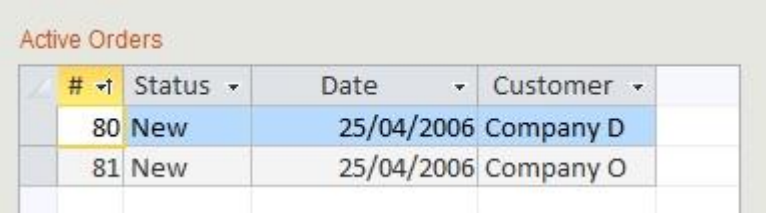


Figure 61 – Freeform Panel "Active Orders"

To define the UI Data Grid object "Active Orders" we will follow these steps:

- Define a View Table "Active Orders":
    - based on the following View Columns (listed as source View Table->View Column for ready identification only); "Order Number" as Orders->Order ID, "Order Status" as Orders Status->Status Name, "Order Date" as Orders->Order Date, and "Customer" as Customers->Company respectively. Note that we only need to select these View Columns and assign them to the View Table - the internal model logic will resolve the appropriate database queries to manage the data based on their pre-defined relationships.
    - Define a basic View Sort "Active Orders Sort" for the "Active Orders" View Table based on the "Order Number" Assigned View Column although we will allow ad-hoc sorting to be enabled and managed by the runtime engine.
    - Finally define a View Filter "Active Orders Filter" for the "Active Orders" View Table based on the condition "AND(Active Orders ->Order Status <>"Closed",Orders->Employee ID=#Current_User_Id)".
- Next, the UI Data Grid Cells must be defined – in this case they are all simple objects that match the corresponding Assigned View Columns of

the “Active Orders” View Table. Note that the contents of each UI Data Grid Cell are defined within separate Freeform Panels to allow maximum flexibility and functionality, although these will be quite simple. For each respective Assigned View Column the corresponding Freeform Panel for the UI Data Grid Cell will be defined as (all set to read only or as an over-ride setting for the UI Data Grid):

- “Order Number” as a UI Text object,
- “Order Status” as a UI Selection “Order Status” which will be based on the “Order Status” Assigned View Column,
- “Order Date” as a UI Text object,
- “Customer” as a UI Text object.
- Note that each of these display objects will need to be “bound” to the respective Assigned View Column to define it as the source of the object’s data. The binding parameter to specify will be “Current” to indicate that data from the currently referenced row of the View Table will be sourced.

- The final step is to define the Double-Click events for the “Order Number” and “Order Date” UI Text objects in their UI Data Grid Freeform Panel definitions to open up an Orders form. However, as we haven’t yet defined this form, we will finalise this later.

The “Active Orders” text will also be set as border text for the “Active Orders” Freeform Panel object. The “Active Orders” UI Data Grid object will be anchored to the top left of the Panel, with resizing options defined for each UI Data Grid Cell to determine the width, plus a minimum height to allow about 10 rows.

The “Active Orders” Freeform Panel will be anchored to the left of the Canvas and its top to the bottom of the “User” Freeform Panel.

9.4.4.1.4 Freeform Panel “Inventory to Reorder”

The “Inventory to Reorder” Freeform Panel also has only one object; a UI Data Grid object to list and access the inventory items that require re-stocking orders. It only lists the Product, Qty Available and Reorder Level – clicking any cell opens up the corresponding Product form for editing.

Inventory to Reorder

Product	Qty Available	Reorder Level
Northwind Traders Boysenberry Spread	0	25
Northwind Traders Dried Pears	0	10
Northwind Traders Curry Sauce	0	10
Northwind Traders Fruit Cocktail	0	10
Northwind Traders Scones	0	5
Northwind Traders Beer	0	15
Northwind Traders Clam Chowder	0	10
Northwind Traders Chocolate	0	25
Northwind Traders Gnocchi	10	30

Figure 62 – Freeform Panel "Inventory to Reorder"

To define the UI Data Grid object "Inventory to Reorder" we will follow these steps:

- Define a View Table "Inventory to Reorder":
  - based on the following View Columns (listed as source View Table->View Column for ready identification only); "Product" as Products->Product Name, "Qty Available" I will discuss shortly, and "Reorder Level" as Product->Reorder Level respectively. "Qty Available" is not yet stored anywhere as data – it is the result of a calculation which fortunately we can also create as a View Column:
    - View Columns based on calculations, or on the basis of a Function as we define them, are calculated in a way similar to repeated spreadsheet functions – any references to other View Columns are resolved via the standard data relationship deconstruction and the appropriate rows of other Assigned View Columns are used as the arguments to complete the current Assigned View Column row's Function value.
    - For the "Qty Available" View Column its value will be determined by subtracting the total purchased from the total sold (for each separate product) based on the records

in the “Inventory Transactions” View Table. This View Column will not reference any other View Column or Virtual Column, it will just define a Function to perform this calculation as:

        - SUM(Inventory Transactions, Quantity, AND(Inventory Transactions->Product_ID = Inventory to Reorder->Product, Inventory Transactions->Transaction_Type = ”Purchased”)) - SUM(Inventory Transactions, Quantity, AND(Inventory Transactions->Product_ID = Inventory to Reorder->Product, Inventory Transactions->Transaction_Type = ”Sold”))
    - Also define a basic View Sort “Inventory to Reorder Sort” for the “Inventory to Reorder” View Table based on the “Qty Available” Assigned View Column although we will allow ad-hoc sorting to be enabled and managed by the runtime engine. Finally define a basic View Filter “Inventory to Reorder Filter” for the “Inventory to Reorder” View Table based on the condition “Inventory to Reorder -> Qty Available < Inventory to Reorder-> Reorder Level”.
- Next, UI Data Grid Cells must be defined – in this case they are all simple objects that match the corresponding Assigned View Columns of the “Inventory to Reorder” View Table. For each respective Assigned View Column the corresponding Freeform Panel for the UI Data Grid Cell will be defined as (all set to read only or as an over-ride setting for the UI Data Grid):
    - “Product” as a UI Text object,
    - “Qty Available” as a UI Text object,
    - “Reorder Level” as a UI Text object.
- The final step is to define the Single-Click events for each cell, for each of the UI Text objects in their UI Data Grid Freeform Panel definitions to open up a Product Details form. However, as we haven’t yet defined this form, we will finalise this later.

The “Inventory to Reorder” text will also be set as border text for the “Inventory to Reorder” Freeform Panel object. The “Inventory to Reorder” UI Data Grid object will be anchored to the top left of the Panel, with resizing options defined for each UI Data Grid Cell to determine the width, plus a minimum height to allow about 10 rows.

The top left of the “Inventory to Reorder” Freeform Panel will be anchored to the right and top of the “Active Orders” Freeform Panel.

9.4.4.1.5 Navigation Panel “Quick Links”

The “Quick Links” Navigation Panel has eight Navigation Panel Item objects, all text icons acting as a ready menu to the main data entry and report objects.

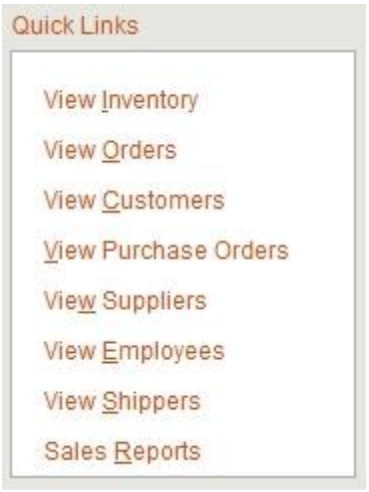


Figure 63 – Navigation Panel “Quick Links”

To define the Navigation Panel we will follow these steps:

- Create each of the eight links as text based Navigation Panel Items, choosing a Vertical layout order with 1 column.
- Each Navigation Panel Item will be aligned to the left border in a list manner. The first will be aligned to the top left of the Panel whilst the last will be aligned to the bottom right of the Panel.
- At this point, do not identify the model objects that will be invoked by actioning any of the Navigation Panel Items – we will define these later.

The top and left of the “Quick Links” Navigation Panel will be anchored to the left and bottom of the “Active Orders” Freeform Panel.

9.4.4.1.6 Freeform Panel "Total Revenue"

The "Total Revenue" Freeform Panel has only one object; a UI Chart object to display a graph of revenue for the currently logged in user. It depicts the monthly totals for the last year as a simple column graph.

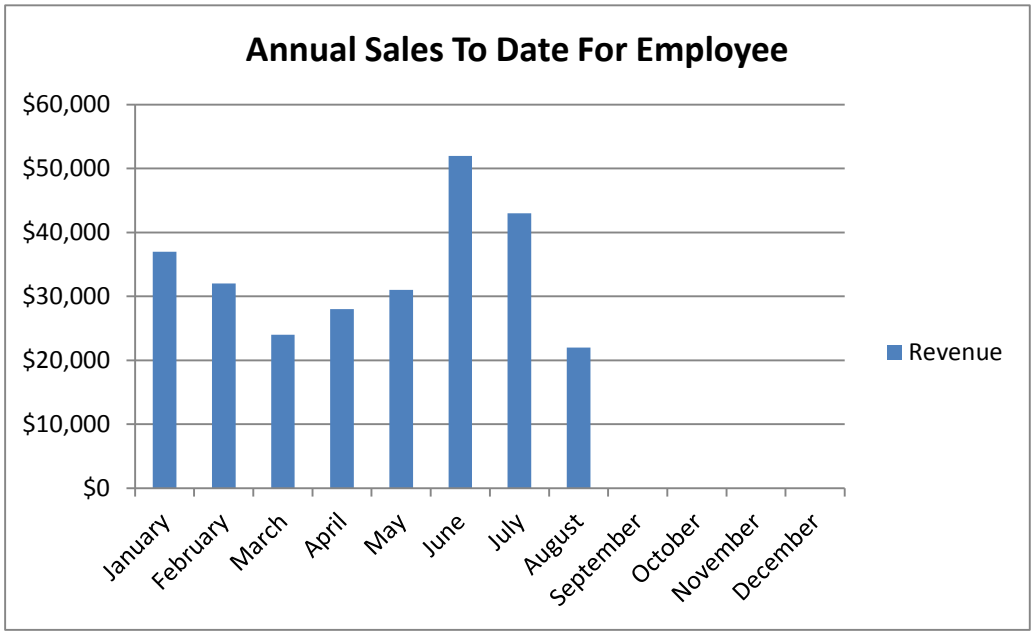


Figure 64 – Freeform Panel "Total Revenue"

To define the UI Chart object "Total Revenue" we will follow these steps:

- Define a View Table "Total Revenue":
    - based on the following View Columns (listed as source View Table->View Column for ready identification only); "Date" as Order Details->Date Allocated, "Quantity" as Order Details->Quantity, "Unit Price" as Order Details->Unit Price, "Discount" as Order Details->Discount plus a new calculated View Column "Revenue" based on the Function "(Order Details->Quantity * Order Details->Unit Price) * (1 - Order Details-> Discount)".
    - Define a View Filter "Total Revenue Filter" for the "Total Revenue" View Table based on the condition "AND(YEAR(Order Details->Date Allocated) = YEAR(#Now), Orders->Employee ID = #Current_User_Id)".

- Define a basic View Sort “Revenue Sort” for the “Total Revenue” View Table based on the “Order Details->Date Allocated” Assigned View Column.
- Finally define a View Group “Revenue Group” for the “Revenue” View Table. The Grouping Level is based on the Function: MONTH(Order Details->Date Allocated), while the Group Value is determined by the Function SUM(Total Revenue->Revenue).

- Next, the UI Chart must be defined – a corresponding UI Chart Type must be selected from available pre-defined types (a simple Column Chart in this case) and the UI Chart Dimensions and UI Chart Series assigned to match those of the selected UI Chart Type. UI Chart Dimensions are the specific number of discrete dimensions that a defined UI Chart Type may utilise – for a simple Column Chart this is just 2 dimensions, one for the X-axis data values plus a second dimension for the Y-axis data values. More complex chart types (e.g. 3D) may have additional UI Chart Dimension requirements. UI Chart Series are the identification of specific axis values representing a series – in this example we have only 1 UI Chart Series for both UI Chart Dimensions however this UI Chart Type does allow multiple UI Chart Series for the Y-axis data points to allow multiple data series to be overlaid. We can also auto-generate multiple UI Chart Series by specifying the generation criteria for the discrete series values as part of the UI Chart Dimension definition. This chart only requires a single UI Chart_Data_Point to be plotted although multiple data points can be defined as required. The UI Chart will be further defined as:
  - The UI Chart_Data_Point will be the Assigned View Column “Revenue”, specifying an aggregation type of SUM.
  - The first UI Chart Type Dimension for this UI Chart Type (a Column Chart) is for the definition of the X-axis data points. We could manually define each of the Months as 12 separate UI Chart Series but instead we will define the rules to auto-generate the Months of the year as part of the UI Chart Type Dimension (for illustration). For this UI Chart Type Dimension we specify:
    - We need a data source that always defines the Months of a year. As a real data source may have incomplete year data

this is not suitable, we will instead generate a View Table "Months" that just has the Months in a View Column. This requires creating and assigning new View Column "Month Number" that will allow an integer entry from 1 to 12 in order, and new View Column "Month Name" and that list the corresponding names from January to December. We first have to create the physical data and then map the Virtual and View structures to achieve this. We could also readily create a View Sort "Financial Year Months" to list this in financial year mode with a Function like "Months->Month Number + IF(Months->Month Number < 7, 6, -6)" if it was required. We select the "Month Number" Assigned View Column as the data source for the auto-generated UI Chart Series.

- We must assign the auto-grouping criteria for the series as a Function in order to extract the correct data, as: MONTH(Total_Revenue->Date) which will then be used to group and match against the auto-generated series values.
- The presentation of the axis label for these UI Chart Series will be a UI Text object displaying the Assigned View Column "Month Name" object. The dimension scale type will be set to LINEAR spacing.

- The second UI Chart Type Dimension for this UI Chart Type is for the definition of the Y-axis data points – we only require a single UI Chart Series here although we could specify multiple series. For this corresponding UI Chart Series we specify:
  - As we have already prepared the key data source that we require to populate the actual series data including the filter criteria, i.e. we want all data to be included, then we only need to specify that this series is to display all remaining data as a logical Yes.
  - The presentation of the data label for this UI Chart Series will be a UI Text object displaying "Revenue".

- The heading for the UI Chart will be a UI Text object displaying "Annual Sales To Date For Employee",

The UI Chart "Total Revenue" object will be anchored to the top left of the Panel. The left of the "Total Revenue" Freeform Panel will be anchored to the right of the "Quick Links" Navigation Panel whilst the top of the Panel will be anchored to the lowest of the bottom of both the "Active Orders" Freeform Panel and the "Inventory to Reorder" Freeform Panel. The bottom of the Panel will be anchored to the bottom of the "Dashboard" Canvas.

9.4.4.2 Canvas "Order Details"

Previously we postponed the links to some of the additional forms and reports objects that are referenced from objects displayed on the "Dashboard" Canvas as we hadn't yet defined all functionality. One of these links is the "Order Details" form which is displayed by double-clicking either the Order # or Date columns in the grid.

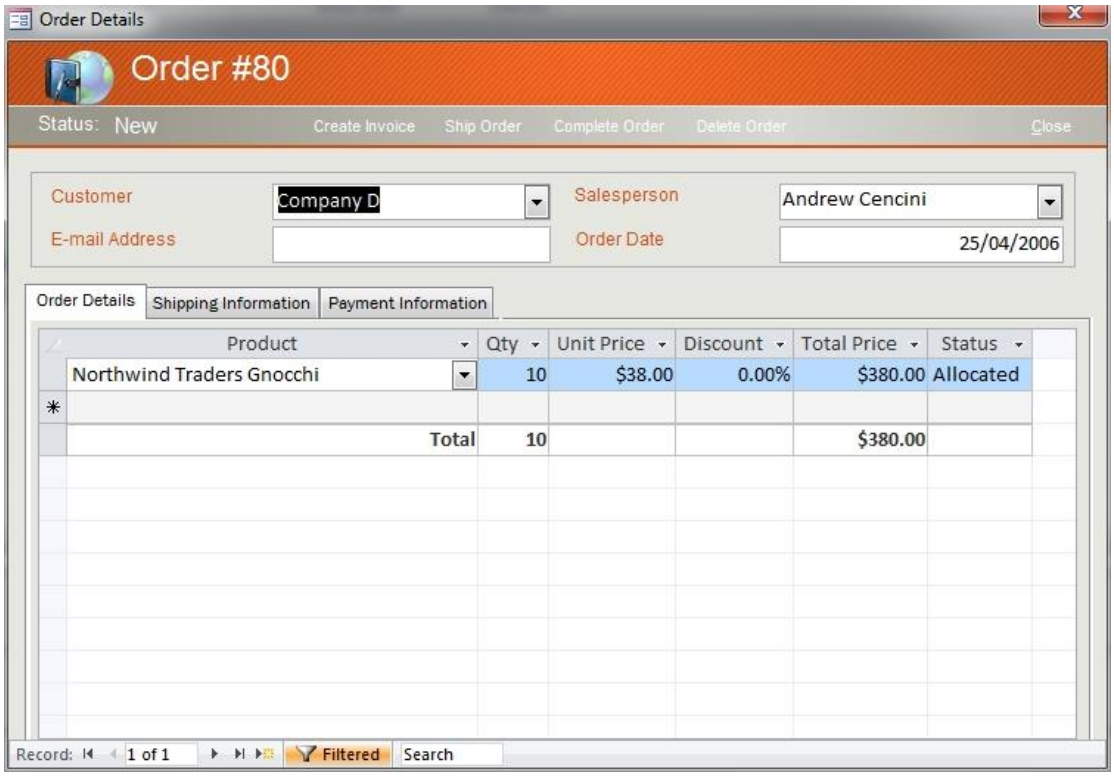


Figure 65 – Canvas "Order Details"

To replicate similar functionality we need to define a Canvas object "Order Details" and populate it with the following panel objects:

- Freeform Panel “Order Header”: to display the current order number,
- Freeform Panel “Order Actions”: to display the status of the order, and some text options to invoke additional order workflows,
- Freeform Panel “Order Info”: to define the primary order details information.

Each Panel will be anchored to the “Order Details” Canvas and other Panels based on appropriate UI Alignment Rules to provide for the minimum, maximum and resizing options as the objects are manipulated by users.

There is another new feature that will be illustrated within this Canvas, the passing of a runtime parameter, in this case the Order Number, to be used as part of Panel initialisation. Any set of objects can be defined as parameters to be passed to another object.

- In order to pre-populate the “Order Details” Canvas object with a pre-selected Order Number, which can occur from previous Panels where the Order is selected from a Grid, we need to check if the Canvas was called with a parameter specifying an Order object. If you recall, we had deferred setting the Double-Click events on the UI Data Grid object “Active Orders” – now we will define them. There are two main actions here:
    - Firstly, we need to define the Application_Event for both objects in the Grid, the “Order Number” UI Text object and “Order Date” UI Text object – this will be a simple Double-Click event that invokes the “Order Details” Canvas object.
    - As we also want the “Order Details” Canvas to display the currently selected Order we also have to ensure that a Pass_Parameter is defined that will provide the selected Order linkage. We do this by defining a Defined_Event Function for each of the above Application_Events and setting this Function to execute before invoking the object event target. The Function will be: SET_PARAMETER(Data_Grid[“Active Orders”].Panel[“Order Number”].Text[“Order Number”], Canvas[“Order Details”], Active Orders[current], null) and SET_PARAMETER(Data_Grid[“Active Orders”].Panel[“Order

Date"].Text["Order Date"], Canvas["Order Details"], Active Orders[current], null) respectively.

- We can now check to see if the "Order Details" Canvas object was invoked with such a parameter by defining an Initialisation event for the Canvas with a Function: IF(GET_PARAMETER(Canvas["Order Details"], Active Orders[OrderNumberVarRow]), OrderNumberVarNum = Active Orders[OrderNumberVarRow].Order Number, OrderNumberVarNum = NULL). i.e. if there was a Parameter of type "Active Orders" then the current row identifier is returned as variable OrderNumberVarRow and we then look up the corresponding "Order Number".
- For completeness we should also then remove this parameter by modifying the above Function to: MULTI(insert existing function from above, REMOVE_PARAMETER(Canvas["Order Details"], Active Orders[OrderNumberVarRow])).
- Before we start to create UI controls that will be bound to the data sources we need to ensure that the initialisation of the Canvas sets the record to either the existing or new Order. The following Function segment should be appended as an additional argument to the above Function: IF(OrderNumberVarNum = NULL, SET_RECORD(Active Orders[New]), SET_RECORD(Active Orders[OrderNumberVarRow])). This now sets the current record of the "Active Orders" View Table to either match the passed in parameter, or to a new row.

This now sets the data source to either a new or existing "Active Orders" record for editing throughout the Canvas.

9.4.4.2.1 Freeform Panel "Order Header"

The "Order Header" Freeform Panel has only a few simple objects; a graphic UI Image plus UI Text objects to display "Order #".

Figure 66 – Freeform Panel "Order Header"

To define the Freeform Panel object "Order Header" we will follow these steps:

- Set the background colour of the Panel to the desired colour. We will also set the text colour of all text objects to white as we define them.
- Define a UI Image object and load the image file to be used, anchored to the top and left of the Panel.
- Define a UI Text object "Order #" with that same text and anchor to the top and left of the UI Image object.
- Finally, we define a UI Text object "Order Number" which is bound to and displays the value of the Assigned View Column "Order Number" and anchors it to the top and left of the UI Text object "Order #".

The top, left and right of the "Order Header" Freeform Panel will be anchored to the top, left and right of the "Order Details" Canvas.

9.4.4.2.2 Freeform Panel "Order Actions"

The "Order Actions" Freeform Panel consists of fairly simple objects; the status of the order, and some text options to invoke additional order workflows – note that we could create a Navigation Panel for these links however we will demonstrate how they can also be achieved as standard object events.

These objects could have been included as part of the previous Freeform Panel however to replicate the background colours it is simplest to just set this as a Panel attribute.

Figure 67 – Freeform Panel "Order Actions"

To define the Freeform Panel object "Order Actions" we will follow these steps:

- Set the background colour of the Panel to the desired colour. We will also set the text colour of all text objects to white as we define them.

- Define a UI Text object “Status:” with that same text, anchored to the top and left of the Panel.
- Define a UI Text object “Display Status” which is bound to Order Status->Status Name, to display the current status of the Order, anchored to the top and left of the UI Text object “Status:”.
- Now create each of the five links as UI Text objects; “Create Invoice”, “Ship Order”, “Complete Order”, “Delete Order” and “Close”. Set the alignment rules to display them as they appear in the above image. Do not set any events for these objects yet except for the “Delete Order” and “Close” UI Text objects where we will define:
    - A Single-Click event to invoke the deleting of the current Order as a simple Function as: DELETE_RECORD(Active Orders[current], Confirm).
    - A Single-Click event to invoke the closing of the “Order Details” Canvas object as a simple Function to invoke the Exit event of the Canvas – this will automatically invoke the Exit events of all child controls within the Canvas and its Panels, as “Canvas.[Name, “Order Details”].run.exit”.

The top, left and right of the “Order Actions” Freeform Panel will be anchored to the top, left and right of the “Order Details” Canvas.

9.4.4.2.3 Freeform Panel “Order Info”

The “Order Info” Freeform Panel has several objects, many relating to the data of the current Order; a group of text and data objects at the top plus a UI Tab control that manages access to the bulk of the Order data. The status of some Order data directly affects the workflows that will be invoked from the UI Text object links in the previous Freeform Panel.

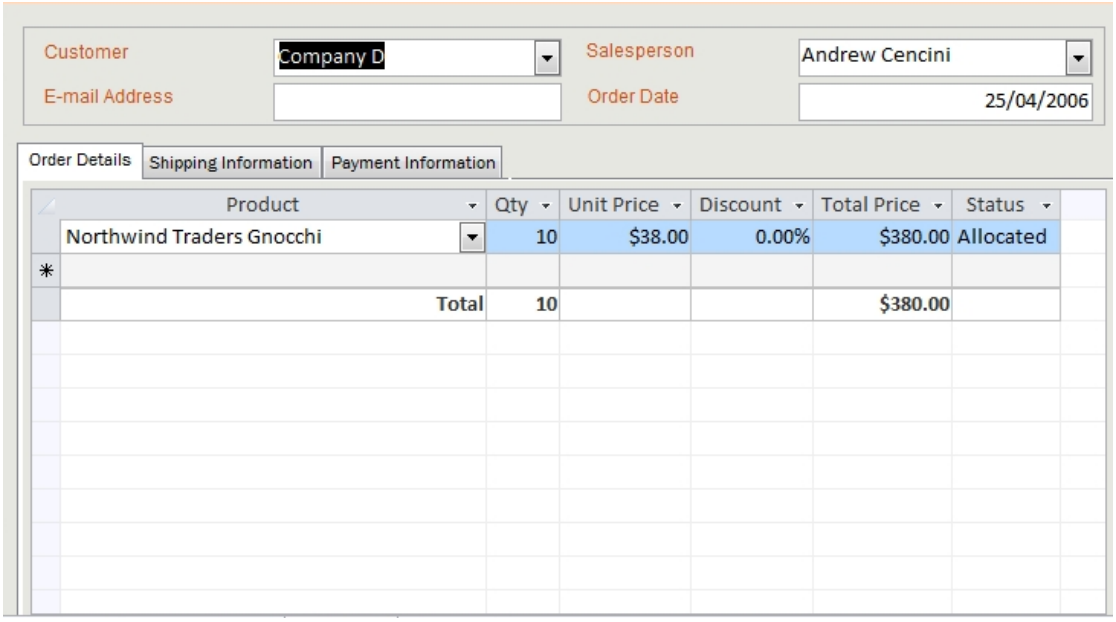


Figure 68 – Freeform Panel "Order Info"

To define the Freeform Panel object "Order Info" we will follow these steps:

- Set the background colour of the Panel to the desired colour. We will also set the text colour of all text objects as we define them.
- Define UI Text objects "Customer", "Email Address", "Salesperson" and "Order Date" with the same text, plus a UI Rectangle to form a cosmetic grouping of the objects. Now create the following UI controls, each bound to the appropriate Assigned View Column:
    - UI Selection "Customer" bound to Active_Orders->Customer, obtaining its list data from Customers->Company. A Change event will need to be defined so that the address details of the company are loaded by default into the shipping address in the "Shipping Information" Tab that we will define shortly.
    - UI Text Box "E-mail Address" bound to Customers->E-mail Address.
    - UI Selection "Salesperson" bound to Employees->Salesperson, obtaining its list data from "Employees" View Table .
    - UI Text Box "Order Date" bound to Active_Orders->Order Date.
- We now need to define a UI Tab object "Order":

- The UI Tab is composed of 3 individual tabs: “Order Details”, “Shipping Information”, and “Payment Information” – each will be defined on their own Canvas as a UI Tab Canvas.
- For the “Order Details” UI Tab Canvas:

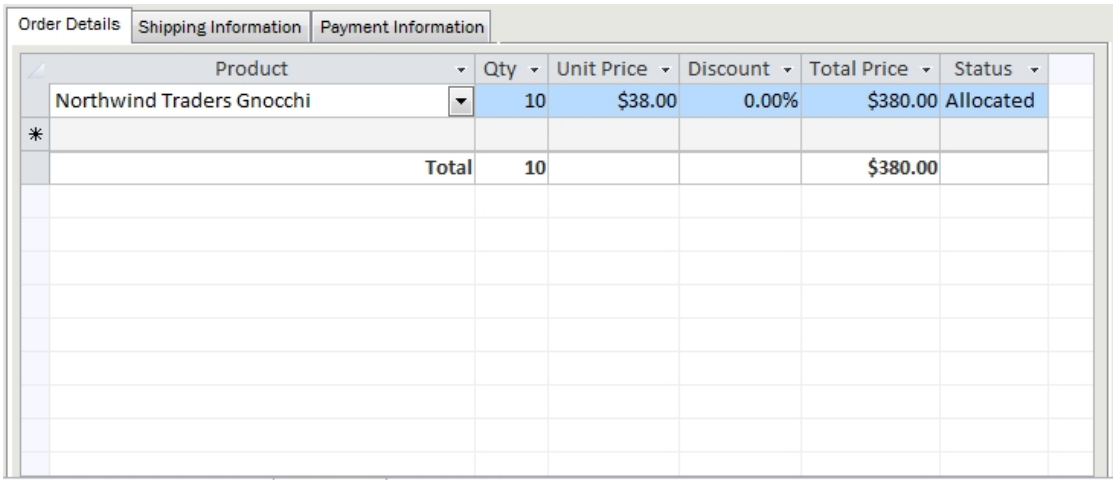


Figure 69 – UI Tab Canvas “Order Details”

- This tab consists of only a single Freeform Panel “Order Details Tab” with a UI Data Grid “Order Details Tab” defined as:
  - We define a View Table “Order Details Tab” with the following View Columns:
    - “Order” as Orders->Order ID (hidden) – this View Column will be set to auto-incrementing as an identifier column,
    - “Product” as Products->Product Name, as a UI Selection initially blank,
    - “Qty” as Order_Details->Quantity, initially 0,
    - “Unit Price” as Order_Details->Unit Price – an Initialisation event Function is defined to preload its value as: IF(Order_Details->Unit Price = NULL, Products->List Price)
    - “Discount” as Order_Details->Discount,

- "Total Price" as a calculated View Column based on the Function: (Order Details Tab->Qty * Order Details Tab->Unit Price) * (1 - Order Details Tab->Discount)
- "Status" as Order_Details_Status->Status Name.
- A Function is required to process each order line for the following business rules:
    - If Qty = 0 then Status = "None"
    - If Qty <= Products->Qty Available then Status = "Allocated"
    - If Qty>Products->Qty Available then ask to create new Purchase Order, if successful then Status = "Ordered"
    - The Change event Function for the UI Data Grid will be: IF(Order Details Tab->Qty = 0, Order Details Tab->Status = "None",IF(Order Details Tab->Qty <= Products->Qty Available, Order Details Tab->Status="Allocated", IF(Prompt("Insufficient Inventory. Do you want to create a purchase order?"),MULTI(Make_Purchase(),IF(Order Details Tab->Qty <= Products->Qty Available, Order Details Tab->Status="On Order")),Order Details Tab->Status = "No Stock")))
    - The above Function uses a system Prompt function to ask for a user entry (with default Yes/No responses). We also define a new

user defined Function Make_Purchase() that invokes the Purchase_Order Canvas with parameters as: FUNCTION(Make_Purchase, Permanent, (), MULTI(SET_PARAMETER(Data_Grid[“Order Details Tab”].current, Canvas[“Purchase Order”], Order Details Tab[current], null), Canvas[“Purchase Order”], REMOVE_PARAMETER(Data_Grid[“Order Details Tab”].current, Canvas[“Purchase Order”], Order Details Tab[current]), ())

- A View Filter “Order Details Tab” is defined to only display Order Details for the current Order, using the Function: Order_Details->Order = Order[current]->Order ID.
- A View Group “Order Details Tab” is also defined that will group the entire table as a single level group allowing a footer definition to sum the total of the Qty and Total Price View Columns – this single level will occur by default as we don’t need to specify a grouping Function or any grouping by View Columns. We do need to specify the View Columns that will have grouping values defined and how the values are aggregated – we will aggregate using Total in the footer for both “Qty” and “Total Price”.

- For the “Shipping Information” UI Tab Canvas:

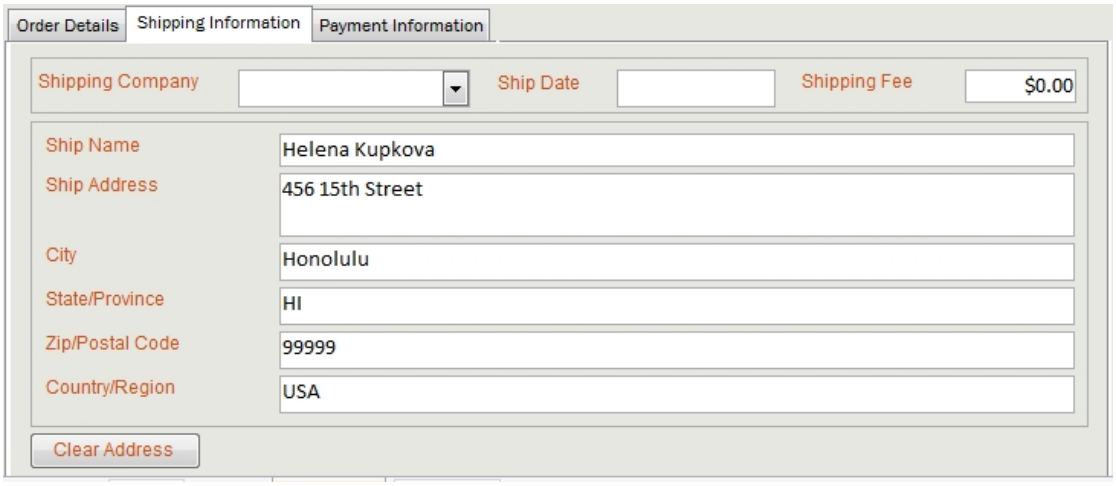


Figure 70 – UI Tab Canvas "Shipping Information"

- This tab consists of three Freeform Panels, the first "Shipping Information Tab - Shipper" defined as: the layout UI Rectangle and UI Text entries (to support) and the View Columns; "Shipping Company" as Shippers->Company as a UI Selection based on Shippers->Company, "Ship Date" as Orders->Shipped Date as a UI Text Box (supplemented by a date picker object) and "Shipping Fee" as Orders->Shipping Fee as a UI Text Box.
- The second Freeform Panel "Shipping Information Tab – Send To" defined as:
    - The layout UI Rectangle and UI Text entries (to support) and the View Columns (as UI Text Box); Orders->Ship Name, Orders->Ship Address, Orders->Ship City, Orders->Ship State/Province, Orders->Ship Zip/Postal Code and Orders->Ship Country/Region.
    - We can now return to complete the Change event for the UI Selection "Customer" that we defined earlier. The Change event will invoke a Function to load the address details of the selected company into the above shipping address objects as: MULTI(Orders[current]->Ship Name = Customers-

>First Name & " " & Customers->Last Name, Orders[current]->Ship Address = Customers-> Address, Orders[current]->Ship City = Customers-> City, Orders[current]->Ship State/Province = Customers-> State/Province, Orders[current]->Ship Zip/Postal = Customers-> Zip/Postal, Orders[current]->Ship Country/Region = Customers-> Country/Region).

- The third Freeform Panel "Shipping Information Tab – Clear Address" defined as a UI Button object "Clear Address" that will clear all of the above address objects when pressed as Function: Freeform Panel.[Name, "Shipping Information Tab – Send To"].run.clear). This is why we grouped these objects into panels, so that a single call could clear all objects, otherwise we would have had to clear each object individually.

- For the "Payment Information" UI Tab Canvas:

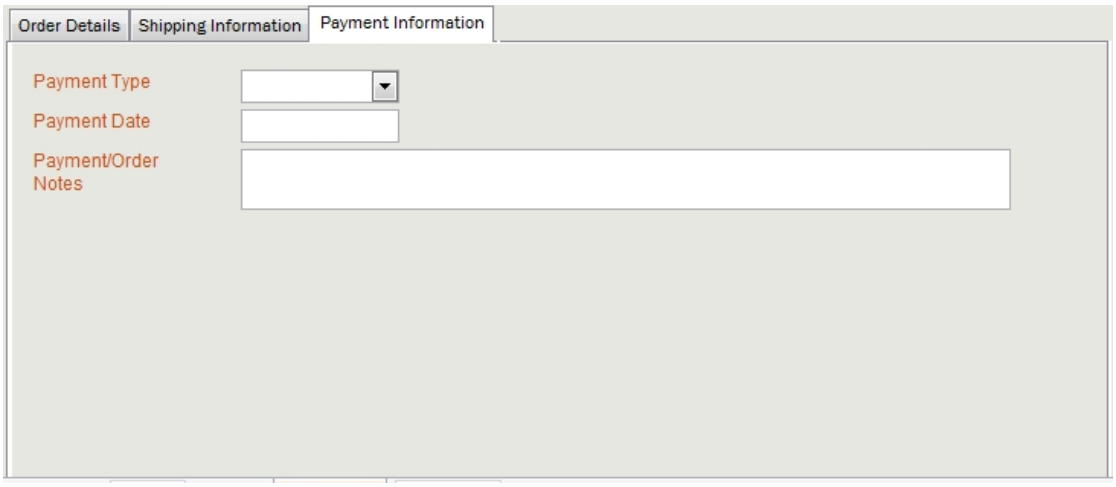
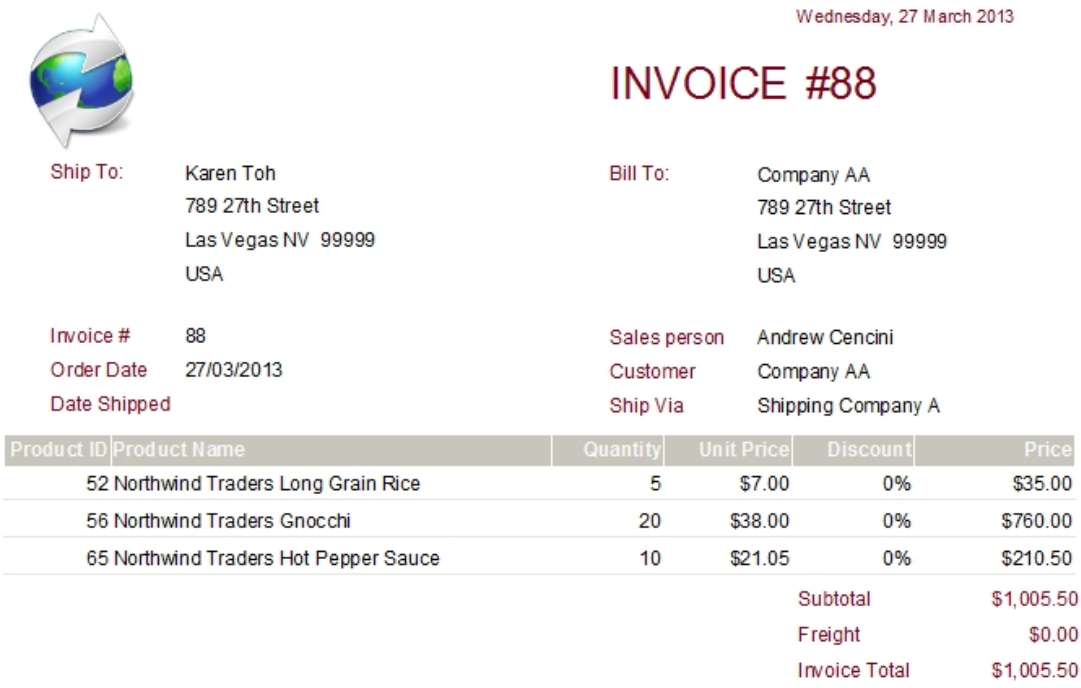


Figure 71 – UI Tab Canvas "Payment Information"

- This tab consists of a single Freeform Panel "Payment Information Tab" defined as: the layout UI Text entries (to support) and the View Columns; "Payment Type" as Orders->Payment Type as a UI Selection based on a list of

(Credit Card, Check, Cash), “Payment Date” as Orders->Paid Date as a UI Text Box (supplemented by a date picker object) and “Payment/Order Notes” as Orders-> Notes as a UI Text Box.

- Earlier, we deferred setting the events for three of the UI Text objects; “Create Invoice”, “Ship Order”, “Complete Order”. We will review the functionality required to finalise these:
    - For “Create Invoice”:
        - We first need each of the following conditions met: there must be; a customer selected, products selected and all must be Allocated, plus the shipping information must be entered. Then an Invoice report is generated, plus each of the Allocated rows is flagged as Invoiced. Subsequent invoices can be re-generated based on any changes.
        - The Function for the click event is: IF(Orders->Customer_ID = null, PROMPT(“Must specify customer name!”,”OK”), IF(OR(Orders->Shipper_ID = null, Orders->Ship Name = null, Orders->Ship Address = null, Orders->Ship City = null, Orders->Ship State/Province = null, Orders->Ship ZIP/Postal Code = null, Orders->Ship Country/Region = null), PROMPT(“Shipping information is not complete. Please specify all shipping information and try again!”,”OK”), IF(COUNT(Order Details Tab, True) = 0, PROMPT(“Order does not contain any line items”,”OK”), IF(COUNT(Order Details Tab, Status <> “Allocated”) > 0, PROMPT(“Cannot create invoice! Inventory has not been allocated for each specified product.”,”OK”), MULTI(Invoice_Report(Orders->Order ID), COLUMN(SET, Order Details Tab, Status, “Invoiced”, ALL))))))
        - Function Invoice_Report() invokes a UI Report “Invoice” based on the passed Order. The UI Report “Invoice” is defined as:



Wednesday, 27 March 2013

INVOICE #88

Ship To: Karen Toh
789 27th Street
Las Vegas NV 99999
USA

Bill To: Company AA
789 27th Street
Las Vegas NV 99999
USA

Invoice # 88
Order Date 27/03/2013
Date Shipped

Sales person Andrew Cencini
Customer Company AA
Ship Via Shipping Company A

Product ID	Product Name	Quantity	Unit Price	Discount	Price
52	Northwind Traders Long Grain Rice	5	$7.00	0%	$35.00
56	Northwind Traders Gnocchi	20	$38.00	0%	$760.00
65	Northwind Traders Hot Pepper Sauce	10	$21.05	0%	$210.50

Subtotal $1,005.50
Freight $0.00
Invoice Total $1,005.50

Figure 72 – Generated Invoice report

- Based on a new View Table "Invoice Report" which extends on the "Order Tab Details" View Table with additional "Order" View Columns, View Sort "Invoice Report" based on the "Product ID" View Column, View Filter "Invoice Report" to select the Function's passed in Order #.
- The following UI Report Bands are defined:
    - **Overall Report Header**: consists of a Freeform Panel "Invoice Header" displaying the displayed header elements from Figure 72 – Generated Invoice report , including the shaded column headings.
    - **Body**: consists of a Freeform Panel "Invoice Body" displaying the 6 data columns lined up to the header column headings.
    - **Overall Report Footer**: consists of a Freeform Panel "Invoice Footer" displaying the calculated totals of the Prices, Freight

and final Total. These can be achieved in a number of ways but the simplest is to create Functions that SUM the View Column values as we have already seen.

- **Overall Report Footer Template**: consists of a Freeform Panel "Invoice Footer Template" displaying the elements for "Page 1 of X". The current and total page numbers are provided by system variables #Current_Page and #Total_Pages. The template report bands are used to overlay common features on every page.

- For "Ship Order":
  - We first need each of the following conditions met: there must be; customer selected, products selected and all must be Invoiced, plus the shipping information must be entered. If the Ship Date is not already entered then today's date is automatically entered. No further changes to customer, products or shipping is permitted
  - The Function for the click event is: IF(Orders->Customer_ID = null, PROMPT("Must specify customer name!","OK"), IF(OR(Orders->Shipper_ID = null, Orders->Ship Name = null, Orders->Ship Address = null, Orders->Ship City = null, Orders->Ship State/Province = null, Orders->Ship ZIP/Postal Code = null, Orders->Ship Country/Region = null), PROMPT("Shipping information is not complete. Please specify all shipping information and try again!","OK"), IF(COUNT(Order Details Tab, True), PROMPT("Order does not contain any line items","OK"), IF(IF(COUNT(Order Details Tab, Status <> "Allocated") > 0, PROMPT("Cannot mark as shipped. Order must first be invoiced!","OK"), MULTI(COLUMN(SET, Order Details Tab, Status, "Invoiced", ALL), IF(Active_Orders->Ship Date = null,

Active_Orders->Ship Date = #Now), Order->Status = “Shipped”, DISABLE(Panel[“Order Info”], Tab_Canvas[“Order Details “], Tab_Canvas[“Shipping Information”])))))

- Note that the above Function could have been simplified if the original application had a better set of Status indicators however as we are replicating the (sometimes poor) design of the current system then we will also replicate similar logic – although even here we could have at least used a Function to reduce the logic replication.

- For “Complete Order”:
  - We first need each of the following conditions met: there must be; customer selected, products selected and all must be Invoiced, plus all shipping information must be entered including the Ship Date, and the Payment Type and Date must be entered. No further changes to the order is permitted
  - The Function for the click event is: IF(Orders->Customer_ID = null, PROMPT(“Must specify customer name!”,”OK”), IF(OR(Orders->Shipper_ID = null, Orders->Ship Name = null, Orders->Ship Address = null, Orders->Ship City = null, Orders->Ship State/Province = null, Orders->Ship ZIP/Postal Code = null, Orders->Ship Country/Region = null), PROMPT(“Shipping information is not complete. Please specify all shipping information and try again!”,”OK”), IF(COUNT(Order Details Tab, True), PROMPT(“Order does not contain any line items”,”OK”), IF(COUNT(Order Details Tab, Status <> “Invoiced”) > 0, PROMPT(“Cannot mark as shipped. Order must first be invoiced!”,”OK”), IF(OR(Orders->Payment Type = null, Orders->Payment Date = null), PROMPT(“Must first specify payment information!”,”OK”), MULTI(Order->Status = “Closed”, DISABLE(Panel[“Order Info”], Tab_Canvas[“Order

Details “], Tab_Canvas[“Shipping Information”], Tab_Canvas[“Payment Information”]), PROMPT(“Order is now marked closed.”, “OK”)))))

The modelling of the “Order Details” Canvas has demonstrated significant capabilities of the MDEIS model including; UI Tabs, data processing logic, complex Functions and UI Reports.

9.4.4.3 Remainder of Application UI Objects

Rather than repeating similar repetitive analysis for the bulk of the remaining user interface objects we will list the major remaining objects and their modelling requirements, focussing only on expanding on any new or novel features:

Create the following UI objects and link them to the associated “Quick Links” Navigation Panel objects:

- “View Inventory” links to a “Inventory List” Canvas composed primarily of:
  - a UI Data Grid “Inventory List” with a Purchase feature that places an order for the associated Product, and
  - a link “Add Product” to a new “Product Details” Canvas that defines new Products.
- “View Orders” links to a “Order List” Canvas composed primarily of:
  - a basic UI Data Grid “Order List”,
  - a link “Add New Order” to the “Order Details” Canvas that we defined in detail previously,
  - a link “View Invoice” to the “Invoice” UI Report that we defined in detail previously.
- “View Customers” links to a “Customer List” Canvas composed primarily of:
  - a basic UI Data Grid “Customer List”, and
  - a link “New Customer” to a “Customer Details” Canvas that defines new Customers. Part of this Canvas are links to “E-mail Customer” and “Create Outlook Contact” – these operations are not part of the core MDEIS model yet can be easily implemented by identifying a web service that can provide the extended functionality, then defining and invoking the web service with the

required data. The “E-mail List” link extracts data to an external file, again not a core function of the MDEIS model but readily performed via a web service call.

- “View Purchase Orders” links to a “Purchase Order List” Canvas composed primarily of:
    - a basic UI Data Grid “Purchase Order List”, and
    - a link “Add New Purchase” to a “Purchase Order Details” Canvas that is used to define, track and pay for new Product deliveries. It also has simple logical workflows defined to manage the transaction components.
- “View Suppliers” links to a “Supplier List” Canvas composed primarily of:
    - a basic UI Data Grid “Supplier List”,
    - a link “New Supplier” to a “Supplier Details” Canvas that defines new Suppliers. It also has links to “E-mail Supplier” and “Create Outlook Contact” that would use the same simple web service solution, and
    - Part of this Canvas are links to “Collect Data via E-mail” and “Add From Outlook” – these operations are not part of the core MDEIS model yet can also be easily implemented by using web services that can provide the extended functionality. The “E-mail List” link extracts data to an external file would also use a web service call.
- “View Employees” links to a “Employee List” Canvas composed primarily of:
    - a basic UI Data Grid “Employee List”,
    - a link “New Employee” to a “Employee Details” Canvas that defines new Employees. It also has links to “E-mail Supplier” and “Create Outlook Contact” that would use the same simple web service solution,
    - Part of this Canvas are links to “Collect Data via E-mail”, “Add From Outlook” and “E-mail List” that would use the web service solution, and
    - a link “Reports” that provides a menu of various UI Reports.

- "View Shippers" links to a "Shipper List" Canvas composed primarily of:
    - a basic UI Data Grid "Shipper List",
    - a link "New Shipper" to a "Shipper Details" Canvas that defines new Shippers. It also has links to "E-mail Supplier" and "Create Outlook Contact" that would use the same simple web service solution, and
    - Part of this Canvas are links to "Collect Data via E-mail", "Add From Outlook" and "E-mail List" that would use the web service solution.
- "Sales Reports" links to a "Sales Reports Dialog" Canvas composed primarily of:
    - Selection criteria controls for; "Select Sales Report" type, "Select Sales Period", "Filter Sales Items" by Product, and "Year, Quarter, Month" period selections,
    - The above selection criteria are passed as parameters to a "Sales Report" UI Report.

This would complete the logic definition of this fairly complex ordering application within the MDEIS framework as a modelled application.

9.4.5 Review of Complex Application Modelling Example

The Northwind application has demonstrated the application modelling capability of the MDEIS framework of many of the simple and complex features of the model.

Commencing with the data modelling is the natural initial point for many application developers, particularly if migrating from an existing database schema. Once the data structures have been modelled to capture the underlying physical data structures (Virtual), the working Views of the schema provide a simple ongoing mechanism as they abstract the users away from the need for structural and transaction issues as the model has captured the relationships and the runtime execution engine will then manage the required transactions.

Once the physical database connections are provided future reverse engineering and application model generation can proceed based on wizards that can be provided as core runtime components or even defined by third parties as standalone MDEIS applications or Functions to generate large portions of the model automatically.

Throughout the Northwind example View Tables were regularly defined on an as needs basis for any specialist data processing requirement, drawing on existing View Columns from any required source, providing local aliasing as desired to simplify local requirements. Transaction management is managed internally although local specification via View Sorts, Filters and Groups with direct update Functions provides a wide range of flexibility to access and modify single data or batches of data.

While a future wizard can extend from the data schema to also generate many of the user interfaces and processing logic, our example relied on manually demonstrating the user interface object definitions. As demonstrated, logic elements will also be asynchronously defined as required in tandem with the definition and placement of user interface objects.

The Northwind example provided extensive examples of the user interface navigation and display objects with many of the complex user interface objects such as alignments, tabs, grids, selections and reports demonstrated.

Logic chains are provided based on links to other objects with additional processing provided by Functions. Simple links such as navigation controls as well as the more complex passing and setting of parameters was demonstrated, allowing the behaviour of Functions or user interface objects to be modified based on their logical workflow source. Functions were often used to provide minor or major processing of logic or data. Further examples of Function capability are provided later in this chapter.

9.5 Conclusion

This chapter presented a detailed demonstration of how the MDEIS framework can be used to capture and model both simple and complex aspects of EIS applications.

The standalone examples such as Functions were provided as capability stepping stones to illustrate the solution of separate logic cases.

The more detailed application modelling walkthrough using the Northwind application and database utilised many of the features of the MDEIS framework in capturing and modelling the application design.

The accompanying description of this application capture and model definition is necessarily quite verbose although is much quicker in a manual capture mode and the

use of the accelerants and editors (see Chapter 7 - Accelerants for the Iterative Design of EIS Models) would have a significant effect on simplifying model definition and capture.

The use of an efficient editor would level the playing field in our assessment – power users can layout user interfaces as easily as technical programmers, and the additional time that programmers use to also capture the structural and coding constraints of application code and additional functionality would provide power users with any additional overhead to complete any required Functions for logical processing.

This seems in line with the original optimisation assumption of merging the analysis and design phases by permitting business analysts and power users to directly and iteratively model the application as the primary application documentation method.

The editor will also more directly enforce discipline and integrity upon the business analysts, power users and users who do modify or extend the application model both through the design and subsequent maintenance phases of the lifecycle. The application of Variant Logic is expected to be a major component of both the ongoing usage and business case payback of the MDEIS application lifecycle.

Chapter 10 - Conclusions and Future Directions

10.1 Introduction

Software development as a whole is a huge worldwide endeavour in terms of cost and effort. The subset of EIS style application development, as the subject of this thesis, is a large, complex and expensive proportion of this effort with major inefficiencies consumed by continuous duplication, re-engineering and updating software applications and customisations.

In the following sections of this final chapter I will summarise the issues faced and my thesis contributions.

10.2 Thesis Overview

Continuing technology advances have not fundamentally altered the outcomes of the software or system development methodologies used to develop Enterprise Information Systems (EIS) style applications. In general, most methodologies still maintain the basic paradigm for system development as some form of the traditional stages of analysis, design, code, test and deploy. While iterative efficiency and

useability improvements have been gained, they have not changed the overall magnitude of the total development effort.

This research project proposes that performance of the analysis and requirements gathering, with efficient collection of this information can also perform the bulk of the design phase for an EIS application, largely as a simultaneous activity, with the collective design requirements stored and available in a suitable model. This research aims to develop a meta-model structure and framework that will allow EIS style applications to be executed automatically from the model with the availability of a set of specific runtime components.

This expectation is based on the highly structured nature of EIS applications that I summarise 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 utilise database transactions in a (usually) relational database environment to complete the actions. As EIS applications are typically structurally repetitive they tend to be a technically simpler subset of possible computer applications. They consist of applications such as logistics, human resource, payroll, project costing, accounting and other general database applications.

The successful outcome of such an approach has the potential to drastically reduce the time to develop and deploy an EIS application when the model based framework is available. The virtual elimination of the coding, combined with the minimisation of the testing and deployment stages would have significant benefits for both the developer and the end users - a benefit that would be further amplified when the lifecycle for new versions of the applications is included.

This thesis addresses many of the problematic issues associated with large-scale software development. The main objective of this thesis is to develop an alternative development methodology by proposing a model standard for defining and producing Enterprise Information Systems in a much cheaper and simpler way, exploring additional benefits that might be derived from subsequent usage of a model based framework.

This thesis achieves its objectives through the following main outcomes:

1) An EIS model structure this is formally defined and that will adequately model the application features required in EIS applications encompassing the user interface, business logic workflow and transaction processing capability.

2) The accelerator mechanisms that has formally deisgned to expedite and simplify population of the model by users, with user specified model data such as rules and relationships between application objects, wizards for model data entry sequences, user interface templates, external model reverse engineering and additional model objects that will facilitate integration between multiple models.

3) A prototype that has developed that could be used to automatically execute the EIS application models. This runtime engine is expected to be service based utilising any combination of technologies and deployment strategies. The high level design will document the key features and attributes of the runtime execution environment.

4) An interface language specification that has formally defined and developed that could be used to access data and application services from external applications. Based on a service-oriented architecture (SOA) all functions of the solution will be available for de-centralised cloud access and integration using common standards.

10.3 Issues Addressed in this Thesis

Repetitive and duplicated software development are a major expense to organisations and a huge effort drain on a global basis. Many modern business users who have grown up with technology on a daily basis also have a higher level of technical savviness as well as business process knowledge that could be better employed to aid in the production of better EIS applications.

This thesis focuses on the presenting a model for EIS applications that would allow the entire application to be modelled, rather than coded, with much of the effort provided by business users, rather than technical software programmers. The key research proposition of this thesis is; simplified development lifecycle, application solutions focussed closer to business needs, greater application flexibility and faster deployment through the use of an associated runtime execution framework that then executes the models directly. In the following section I outline the problem areas that I have identified.

10.3.1 Research Issue 1: The Definition of an EIS Model Structure

The definition of a model structure that will adequately model the application features required in EIS applications encompassing the user interface, business logic workflow and transaction processing capability. Another key difference here is the addressing of the entire application structure, plus model development and deployment version control.

The great majority of modelling effort is aimed at either specific application layers, and most restrictively, targeted to the usage of highly skilled technical software coders. With the entire application logic captured in a model, and with the associated runtime execution engine, the application model can be directly executed by the users.

The model also supports users to be assigned the capability to be logic definers, editing existing application model logic or creating new segments of the model. Many business users are already fluent in many logic definition functions due to their skills in commonly available business database and spreadsheet applications and increasingly in conditional logic processing of events on smartphones, tablets and personal computers. This direct logic definition capability has major potential in speed of solution delivery, cost reduction and accuracy of process capture.

A major aspect of the model is the definition of Variant Logic, the capability for alternate logic streams to be defined to supplement or replace core logic. This ability to allow business users to define or redefine the EIS application (within assigned constraints) allows ultimate flexibility and safe customisation of personal logic that can truly optimise the local workflows of users and groups.

As all application logic is captured within the model, including model elements, workflows and visual structures, the model can also be exported into other human interpretable formats such as structured documentation and training manuals. A major benefit is that there never need be an outdated manual, as the latest version can generated on demand from the current model structure.

Such documentation can also be generated on a role or personal basis by selecting only the model objects that particular users, roles or groups have access to, providing highly targeted, specific and relevant information to users. Such targeting can also include defined Variant Logic ensuring synchronisation with their personalised or otherwise modified logic execution.

The automatic version control of the application model supports a full temporal execution mode of the application model. Whilst full audit tracking of data is a well understood capability and implemented in some application systems to maintain a full historical record of all data transactions and changes, these systems can become limited (and expensive to maintain) as new versions of the application and database schemas are progressively implemented. As the application model is itself data, the application model is itself subject to similar audit tracking (as to the data) that thus provides a full and accurate matched temporal application execution capability, regardless of model (application) updates.

The model version control also supports an automated model update capability. Instead of replacing compiled software modules, application model updates are simply applied streams of application model changes. Deployment testing can also be greatly reduced as precise identification of all changes can be clearly communicated to users. Supporting testing documentation can also be precisely generated to match only those tests that are required by the corresponding logic changes.

Variant Logic (customisations) do not always have to be re-engineered for compatibility as often happens in current EIS applications – when there are no logic collisions there is no need for action. It is possible that some core application model updates may then trigger potential conflicts in some user-defined Variant Logic. Where this does occur, the precise nature of the logical conflicts can be readily identified to the Variant Logic definers so that any subsequent re-definition of the Variant Logic to re-establish compatibility with the new core application logic can be performed with comparative ease. There is no risk of being able to execute an incompatible Variant Logic.

The ability to manage application models down to individual logic segments presents another unexpected opportunity for distributed execution environments. Where an identical or similar (enough) application model is executed as discrete instances, and possibly as part of a large distributed structure of instances, various direct inter-instance executions can be provided. I call these Distributed Execution Requests (DER) and have currently defined the following types:

- **Data Replication**: defines the automated transfer of transaction or summary data between application model instances.

- **Key Authorization**: defines a distributed schema for obtaining key, identifier or sequence based data from a pseudo master application model instance simulating a distributed authorization hierarchy or other virtual topology of application model instances.
- **Logic Variant**: defines the transferring of a locally defined Logic Variant to other application model instances for local execution.
- **Workflow Trigger**: defines a pseudo master application model instance to automatically escalate defined application workflow objects requiring transaction authorization beyond local authorization limits.

When implemented throughout any organisational instance topology of application model instances, these DERs can provide a high level of automated organisational integration without the need for any customisations.

10.3.2 Research Issue 2: Design Accelerants for the Iterative Design of EIS Models

A key objective of this thesis was to shift the main effort of application development requirement for the EIS application logic from technical programmers to application users, with the greater business logic complexity emphasis on power users and business analysts. This shift in effort focus and required expertise necessarily changes the basic application development lifecycle, and believe will greatly simplify it.

Analogies will still exist for the modelling processes along with some unique aspects as facilitated by the use of common meta-data modelling. Defining application model meta-data broadly falls into a combination of the following:

- **Defining new meta-data**: creating new meta-data definitions for the modelled application,
- **Deriving the meta-data**: from some existing non meta-data EIS application based objects such as reverse engineering from existing database schemas,
- **Editing existing meta-data**: to modify existing aspects of a model or extend the application logic,

- **Merging meta-data models**: where multiple meta-data EIS application models exist their meta-data models and thus application functionality can be merged.

The initial benefit would be a merging of the analysis and design stages, as the capturing of the application requirements into a suitable design metaphor, such as the proposed model editor, will directly facilitate model execution. The model editor will facilitate definition of every aspect of the application logic plus define wizards that will prompt for and auto populate the logic definition for the more common logic definitions of; menus and navigation, data structure, canvases (forms), data grids, workflow sequences, and reports. Reverse engineering from data schemas will provide major accelerants – a well-structured data schema could allow the wizard based generation of entire applications as working prototypes.

In many instances, a development stage will not be required, where the provided and supported modelled functionality is adequate, although some specific requirements may require the development of complex logic or even third party integration.

As application models are progressively enhanced, third party providers can develop compatible application models similarly to the current markets for third party applications. Additional toolsets can also be provided that provide complex logic and potentially model extensions.

Merging application models becomes an option analogous to merging the source code and features of two separate traditional applications, although the model based nature of the merging provides opportunity for further optimisation. I identified the following areas of model merging that can be applied to immediately and progressively integrate the functionalities of two separate application models, with comparatively minor effort compared to traditional application re-development:

- **Standard Object Referencing**: the simplest meta-data merge option involves creating new references in one meta-data model to existing objects in a second meta-data model to provide access to the application features of the second meta-data model to users of the first meta-data model. This provides a visual object level integration only.
- **Virtual Data Object Mapping**: provides deeper level model merging and integration of similar meta-data objects between multiple models that

effectively achieves a rationalisation of the underlying relational data structures. This provides an underlying data level merging between the application models.

- **Object Envelopment**: allows defining an object from one meta-data model as a virtual instantiation of a similar object from the other meta-data model, effectively replacing an object and reducing potential duplication between meta-data models. This is the most complete form of model merging.

A reduced testing stage can be employed as only the modelled semantic logic may require testing rather than the usual case of all syntactic logic. The model object version control can provide precise definitions of the model objects that have been modified which allows targeted testing of only the new logic segments. This rationale can apply to local model changes, Variant Logic, as well as to updated core logic model elements.

The deployment stage can then largely be eliminated as the model updates can be deployed automatically as the individual model changes and updates that have been made to the application model. This update can be performed on a sequential basis (matching the temporal order they were originally performed) or on an analysed version differential basis on an object basis.

10.3.3 Research Issue 3: Design of a Prototype Agile Platform for Dynamic Execution

The domain specific model for EIS applications, as defined in this thesis, requires a separate execution environment that transforms the model into operational use. Generally, you would expect that the model editors would maintain the application models in a verified state, however, as there is potential for a variety of model update mechanisms, prudence would dictate that the runtime engine for the application model would also verify the integrity of the defined model prior to invoking the matching executable functionality for all modelled elements.

The general requirement for any runtime engine (or components) is that full compatibility with and support for all features of the meta-data EIS application model is maintained, ensuring that the same model can be executed by any individually architected runtime engine (or components) and process the inputs to obtain identical outputs.

A runtime engine that was based on some form of intermediate (but automated) compilation from the source application model logic segments to present the user with the appropriate execution environment would technically satisfy the basic requirements of the runtime environment. However, a key desirable aspect of the framework is to provide users with a real-time dynamic execution capability for any discrete model change, thus allowing logic definers with an immediate define and test feedback for efficient logic development. Thus the core requirement is that the current meta-data EIS application model must be the direct source for the runtime engine, minimising any obvious or convoluted compilation processes, as well as avoiding any manual or delayed deployment of executables, particularly when Variant Logic (customisations) have been defined by the end users.

The architecture is expected to be service based and support general cloud based access. In addition to supporting all defined model logic operations, the runtime engine must support the underlying model based advances of; temporal data and model meta-data execution with associated transaction roll-back and roll-forward, Distributed Execution Requests between distributed application model instances, and model merging data and object rationalisation.

10.3.4 Research Issue 4: Definition of an Interface Language Specification for Universal Cloud Access

The stored meta-data model is the entire basis for the definition and subsequent execution of the meta-data EIS applications. Much of the application logic workflow relies on the relationships and links between the visual objects defined as the user interface objects. However, there is typically the need for additional logical processing definition beyond visual object relationships, and the limited capabilities of induction and deduction of the data schemas that can be provided by reverse engineering.

Additional command structures are defined to communicate direct instructions to the meta-data EIS application runtime engine and its layers, to both define new meta-data components and to execute meta-data components in response to defined logic. These commands are common to but accessible through the two key system interfaces; user logic definers requiring a simplistic means to define both simple and complex logic sequences, as well as a service based interface language specification that is used to access data and application services between modules or components of

the runtime execution engine, between instances or application model instances, or between third party systems requiring some service integration.

In order to satisfy providing logic definition capability to business users, the function metaphor common to millions of existing users has been extended. Based on the functions that are commonly available in the major spreadsheet programs in widespread usage, and either extending their definition or defining new functions for specific application model operation features, a comprehensive yet relatively simple logic definition capability has been developed.

Functions have been defined that provide; model object manipulation (creation, retrieval, update, delete), system level definitions, data management, logical and conditional processing, group data analysis, as well as the common date, time, mathematical and text based processing.

Additional specialised functions have been defined for; application model management, user defined functions, Variant Logic, temporal execution management, Distributed Execution Requests, application update and rollback, transaction management and security management.

All underlying model execution transactions between any application model based instance modules or components and any other similar or diverse environment will utilise the same commands as services to fulfil the required executions, providing full execution potential through the cloud to any user, service and system combination subject to the defined security limitations of logic definers, execution access and distributed execution access.

The combination of open access to logic definers, instead of restrictions based solely on technical knowledge and capability, as well as open access to application logic features, instead of restrictions based on commercial or architecture considerations, is a major facilitator to; reduce application definition costs and timeframes, increase personal user productivity and simplify application integration.

10.4 Solution Development in this Thesis

In **Chapter 1 -** , I confirmed that EIS style application software development has not generally received the magnitude of benefits expected by the variety of project management methodologies and systems development lifecycles and methodologies that have been attempted. While applying appropriate project management and

following a suitable systems development lifecycle can clearly provide major benefits to good teams, the rates of failure still remain high while the overall lifecycle efforts have not greatly altered due to the fundamental to require the ongoing availability of technical programmers to facilitate all ongoing logical and technological based application changes.

In **Chapter 2 -** , I concluded that whilst there are many commercial products and ongoing research into aspects of modelling application development, the vast majority were focussed on specific layers of an application model rather than an entire application model. Further, the general target users of these tools is for technical programmers rather than business users empowerment. A model and execute style solution to develop EIS style applications can be used more directly by knowledgeable business users to directly capture their requirements and generate an application without the need for technical program coders, requiring only a simplified development lifecycle and greatly reduced overall effort.

In **Chapter 3 -** , I analysed the core problems and defined four key research areas. The **first** is to develop an EIS Model Structure - by capturing the business requirements into a model from which the application would then be directly executed could produce applications much faster with much greater flexibility. The **second** is to Design Accelerants for the Model – all models require an efficient means of populating the model, requiring an EIS model editor that is primarily used directly by such business users, to simultaneously capture their requirements into an application design metaphor, as well as the use of reverse engineering, wizards and templates for common workflows and user interactions. The **third** is to Design a Prototype Runtime Engine – I provide a high level design for the runtime engine, to execute the EIS model (which has captured all of the application's logic requirements), including special features that can be uniquely provided by the use of a source model rather than fixed source code. **Finally**, to specify a Cloud Access User Language – to further provide for specifying additional business logic by defining a function based language similar to that used in major spreadsheet software, and already familiar to millions of business users – also accessible as web services for remote and cloud based interfacing, integration and execution. Associated aims were also defined for each research area as well as an in-depth simulation.

In **Chapter 4 -** , I have described the key design requirements and capabilities of the temporal meta-data framework and how it is used to first record the design of the

EIS application in its meta-data structure, and then directly execute the meta-data EIS application from the meta-data with the runtime engine of the temporal meta-data framework, with no direct coding required, potentially achieving greatly optimised and reduced development efforts. Business users can be provided with the ability to modify and specify their own application functionality within the meta-data EIS application without the need for specialist technical development. The framework would be applicable to the wide range of business EIS style applications.

In **Chapter 5 -** , detailed models are developed to address each of the key requirements of the framework, demonstrating how the meta-data EIS application can provide the fundamental advances in providing the closer integration between EIS applications and the business environment. The model supports the ability for users to define their own application workflows for both process flow and authorisation provides the opportunity for personal workplace optimisation as well as the security of adequate authorisation. The developed Variant Logic concept is perhaps one of the highest potential optimisers, allowing users to customise any aspect of the defined meta-data EIS application. This empowerment option to users to develop or modify alternate or supplemental application logic to best fit their own processes yet maintaining compatibility within the organisational environment offers almost limitless versatility.

In **Chapter 6 -** , I have described the design of a prototype architecture and runtime execution engine model for the meta-data EIS framework that can implement the modelled features. The design overview addresses the major conceptual design models clarifying an overall execution architecture for the meta-data EIS applications that can also support a maximal mobile user base via global cloud based services. The advanced features of the meta-data EIS application model framework and their associated execution requirements have also been clearly expanded from the base design models. The stated architecture will also utilise secure web services as the inter-module and inter-instance interface standard, permitting alternate platform modules as required, and supporting global cloud access to secured model objects wherever they are available and required.

In **Chapter 7 -** , I have provided the capability to populate and define the meta-data EIS application models for execution by the runtime engine. I described the options available for easily defining the meta-data EIS application models with a greater concentration on how to create and use a GUI based meta-data editor, denoted

as an Integrated Meta-Data Modelling Environment (IMDME). I also explore an interesting option for creating the IMDME editor is to utilise a recursive development process whereby we initially hand-code a basic version of the IMDME editor as an example of a meta-data EIS application and then use the first version of this executing meta-data EIS application to then more easily define meta-data and thus further functionality for the next IMDME editor version. By combining this approach with the creation of purpose meta-data defining wizards (themselves as meta-data model instances), and with any specialist user interface objects to aid in visual data modelling or workflow style tasks, a comprehensive IMDME editor can be progressively created. I also presented multiple model merging options that simplify how multiple meta-data EIS application models can be readily merged together to provide a single cohesive larger EIS application. The options presented need only manipulate the meta-data model objects rather than requiring any wholesale redevelopment of entire applications or modules as occurs in traditional application development.

In **Chapter 8 -** , I described how the MDEIS framework design metaphor is addressed at business analysts and power users, who are often very familiar with both the use of spreadsheets and functions, and the fundamentals of relational data structures, the use of often similar Functions provides almost instant familiarisation with many features. This objective would also reduce the learning curve to allow many normal business application users to progressively tweak the application logic and modify as Variant Logic to provide additional localised optimisations, each with the potential to be made available to other local and distributed users. While much of the overall structure and fundamental data transactions of an application can be deduced and inferred from a well-constructed data schema the finer details and major data processing logic require additional logic to be defined. In the meta-data application lifecycle these stages can largely be collapsed into a single stage as the analysts can capture the requirements directly into the meta-data model as both a documentation and simultaneous prototyping platform. By allowing secure access to all Functions and features of the MDEIS model via web services also promotes layer and module separation of the runtime engine plus allowing universal access to and from other remote application and database systems whether executing legacy or MDEIS technologies.

In **Chapter 9 -** , I presented a detailed demonstration of how the MDEIS framework can be used to capture and model both simple and complex aspects of EIS applications. Starting with basic standalone examples such as Functions as capability stepping stones to illustrate the solution of separate logic cases, then up to a more detailed application modelling walkthrough using Microsoft's Northwind Order Management System application and database, utilising many of the features of the MDEIS framework in capturing and modelling the application design. Our detailed description of this application capture and model definition is necessarily quite verbose although would be quite fast when the full editor, framework and accelerants were available. The use of an efficient editor would level the playing field in our assessment – power users can layout user interfaces as easily as technical programmers, and the additional time that programmers use to also capture the structural and coding constraints of application code and additional functionality would provide power users with any additional overhead to complete any required Functions for logical processing. This seems in line with the original optimisation assumption of merging the analysis and design phases by permitting business analysts and power users to directly and iteratively model the application as the primary application documentation method.

10.5 Thesis Contributions

Large scale software development such as EIS applications is an evolutionary complex task, often cobbling together segments of useful legacy code from a variety of technologies. Implementing customisations for customers can be very expensive for customers, due to commercial considerations of vendors as well as the practical issues behind maintaining localised code for one off purposes.

Modelling EIS applications and executing the models directly within a runtime execution environment can provide the sort of paradigm shift that can drastically change the current widespread inefficiencies of repetition and duplication that the global EIS application user and developer community currently face. I have however found very few examples of EIS application style modelling environments that address the entire EIS application modelling space, and none that are targeted towards business users rather than more technical coding staff.

Further, adopting a model based approach also offers a number of advanced application features that can be directly delivered due to the atomic object management basis of the model over the unmanaged nature of typical application source code.

10.5.1 Contribution 1: Comprehensive EIS Model Structure

The definition of a model structure that will adequately model all application features and logic required in EIS applications addressing the entire application structure, permitting the model to be used as the sole basis of execution by the associated runtime execution engine.

Additional model features have been included that support advanced execution modes that have been identified, many that are unique to a model based execution environment; Variant Logic, temporal execution, automated model update, Distributed Execution Requests, model merging.

10.5.2 Contribution 2: Business User Logic Definers

Traditional application software development often requires multiple roles and skillsets; analysts, designers, programmers, testers, trainers, system engineers, each with specific toolsets and techniques. Each integration between the roles requires interpretation skills and presents opportunities for errors in capture and translation of requirements.

By providing a model and tools that can largely be used by business users, organisations will quickly become empowered with a ready workforce that can contribute to the definition of major aspects of the model based EIS applications, as logic definers. Utilising these existing and cost effective logic definers can more rapidly define the required business logic (and thus the application) without the extended delays and costs of requiring often more expensive technical programmers as well as the additional tasks of re-developing the supporting application logic infrastructure for each application (which is otherwise generated directly by the runtime execution environment).

10.5.3 Contribution 3: Simplified Development Lifecycle

Whilst there is great variety between different development methodologies, traditional application software development also requires specific multiple phases or

stages similar to; requirements, design, development, testing, deployment. With a model based EIS application environment, this lifecycle can be simplified and reduced.

Simplified by; combining requirements with design as the model captures these elements simultaneously; reducing or eliminating any development, minimising testing to only new or changed model elements; and all but eliminating deployment by utilising an automated model update.

Additional major effort reductions are provided by the elimination of coding work involved in defining supporting code infrastructure around the basic logic and workflow requirements, as these are provided directly by the runtime execution environment. Any platform specific requirements are handled by a compatible runtime execution environment.

10.5.4 Contribution 4: Variant Logic

Variant Logic represents how application customisations can be provided for in a model based environment. Instead of engineering and maintaining customisations throughout the lifecycle of the EIS application as fundamentally separate application developments, Variant Logic provides the capability for alternate logic streams to be defined to supplement or replace core modelled application logic.

The use of Variant Logic allows business users, acting as logic definers, to define or redefine aspects of the EIS application (within assigned constraints) allowing virtually any logic to be modified or defined. Variant Logic can provide absolute flexibility and safe customisation of personal logic on a scale that dwarfs the potential of traditional customisations.

Currently, application customisations are unique software development projects that are often very expensive hence are often used very selectively as an organisation can afford. Comparatively, Variant Logic can be performed by business users directly to optimise the local workflows of users and groups to any degree. It is not difficult to anticipate that an organisation able to afford only a few customisations could alternately readily justify hundreds or even thousands of Variant Logic, each tailored to provide a direct business optimisation.

10.5.5 Contribution 5: Application Generation Wizards

Application generation is often very repetitive, with application developers often having to proceed through very similar coding exercises for similar types of output, e.g. data entry forms for each data table. The use of pre-defined wizards within the model editor will simplify similar tasks in logic definition of every aspect of the application.

Wizards will prompt for and auto populate the logic definition for the more common logic definitions of; menus and navigation, data structure, canvases (forms), data grids, workflow sequences, and reports. Reverse engineering from data schemas will provide major accelerants – a well-structured data schema could allow the wizard based generation of entire applications as working prototypes.

10.5.6 Contribution 6: Application Logic Merging

Merging the source code and features of two separate traditional applications is complex, particularly if each code base utilises different technologies. Merging application models is analogous to much simpler and can be applied immediately to simply integrate the functionalities of two separate application models with increasing options of integration complexity available.

Simple merging of the models' user interface objects can be readily provided as Standard Object Referencing – a simple but powerful means to merge functionality. To integrate similar data objects I use Virtual Data Object Mapping which effectively achieves a rationalisation of the underlying relational data structures as a combined data source. Object Envelopment can be used to automatically instantiate the use of a data object whenever a similar data object has been defined, providing complete integration of all its workflows.

The model merging options can provide unprecedented integration options between application models, often where using traditional means it would not be possible or only with great additional effort and expense.

10.5.7 Contribution 7: Auto Generated Training

As all application logic is captured within the model, including model elements, workflows and visual structures, the model can also be exported into other human interpretable formats such as structured documentation and training manuals. A major

benefit is that there never need be an outdated manual, as the latest version can generated on demand from the current model structure.

Such documentation can also be generated on a role or personal basis by selecting only the model objects that particular users, roles or groups have access to, providing highly targeted, specific and relevant information to users. Such targeting can also include defined Variant Logic ensuring synchronisation with their personalised or otherwise modified logic execution.

10.5.8 Contribution 8: Runtime Execution Framework Design

The EIS application model structure requires a runtime engine (or components) with full compatibility and support for all features of the meta-data EIS application model. Platform specific requirements of some components will need to be engineered into platform specific versions of the runtime engine components as required e.g. a smartphone version of a user interface renderer vs a desktop personal computer version of same.

The core requirement is that the current meta-data EIS application model must be the direct source for the runtime engine, minimising any obvious or convoluted compilation processes, as well as avoiding any manual or delayed deployment of executables, particularly when Variant Logic (customisations) have been defined by the end users.

10.5.9 Contribution 9: Temporal Execution

The ability to provide an unlimited execution and transaction history across the entire application lifecycle would rarely be attempted by application developers. Whilst it is a relatively simple feature to implement for data, maintaining a full set of compatible executable programs across multiple version upgrades and potentially multiple technology platforms is a huge task. This task is directly solved with the use of a model based application environment.

As the application logic is represented by a model which is data that can be temporally managed as per any other data, a fully synchronised temporal execution capability can be directly provided. Any time point through the applications history can be recovered to the exact state of application logic version and data, albeit in a read only state (initially anyway). Temporal roll-back and roll-forward functions can be used to track through transactions and their impacts.

The use of temporal data management has traditionally been storage expensive which has limited its application to specific application audit requirements. However, as data storage reduces, and with a practical solution to temporal execution provided by a model based application environment, the feature can become readily available rather than virtually impossible.

10.5.10 Contribution 10: Cloud Accessible Services

All commands are common to and accessible through the two key system interfaces; user logic definers requiring a simplistic means to define both simple and complex logic sequences, as well as a service based interface language specification that is used to access data and application services between modules or components of the runtime execution engine, between instances or application model instances, or between third party systems requiring some service integration.

The function metaphor common to millions of existing users of major spreadsheet programs has been extended to provide a comprehensive yet relatively simple logic definition capability for business users. Additional functions have been defined that provide; model object manipulation (creation, retrieval, update, delete), system level definitions, data management, logical and conditional processing, group data analysis, as well as the common date, time, mathematical and text based processing.

Additional specialised functions have been defined for; application model management, user defined functions, Variant Logic, temporal execution management, Distributed Execution Requests, application update and rollback, transaction management and security management.

All underlying model execution transactions between any application model based instance modules or components and any other similar or diverse environment will utilise the same commands as services to fulfil the required executions, providing full execution potential through the cloud to any user, service and system combination subject to the defined security limitations of logic definers, execution access and distributed execution access.

10.5.11 Contribution 11: Automated Update

Many applications and operating systems are provided with an automated update capability, primarily based on tracking the versions of software components and

replacing them with newer versions of the software module. This is a very useful feature although necessarily coarsely implemented at the module level.

For EIS applications, due to the legacy nature of many components, their configuration requirements, as well as the need to often re-engineer any local customisations, automated update is rarely an option for any but the most minor of updates or vanilla of implementations.

As model based EIS applications provide version control down to the atomic level of definition, down to the individual model objects, true automated updates to the model can be precisely controlled and applied as streams of application model changes. Deployment testing can also be greatly reduced as precise identification of all changes can be clearly communicated to users.

Variant Logic (customisations) do not always have to be re-engineered for compatibility as often happens in current EIS applications – when there are no logic collisions there is no need for action as the Variant Logic maintains its original compatibility, which is fully verifiable. If a potential conflict in a Variant Logic is identified, the precise nature of the logical conflicts can be readily identified to the Variant Logic definers so that any subsequent re-definition of the Variant Logic to re-establish compatibility with the new core application logic can be performed with comparative ease, significantly reducing the lifecycle costs of model based customisations (Variant Logic) compared to their traditional counterpart.

10.5.12 Contribution 12: Targeted Deployment Testing

Significantly reduced deployment testing can be achieved within a model based environment as only the modelled semantic logic may require testing rather than the usual case of all syntactic logic including supporting infrastructure code. Further, the model object version control can provide precise definitions of the model objects that have been modified which allows for targeted testing of only the new or changed logic segments. Supporting testing documentation can also be precisely generated to match only those tests that are required by the corresponding logic changes.

10.5.13 Contribution 13: Distributed Instance Integration

Integrating even identical traditional applications can be further examples of expensive customisations. In a model based execution environment, identical or

similar (enough) application model instances can be directly integrated without coding.

These Distributed Execution Requests can be established by authorised users to provide inter-instance integrations such as: automated Data Replication of transaction or summary data; obtaining key, identifier or sequence based data from a pseudo master source as a Key Authorization; transferring locally defined Logic Variant to other instances; and automatically escalating defined application workflow objects to alternate authorisation instances as Workflow Triggers.

These automated integrations can be established between any pairs of instances in any organisational instance topology of application model instances, without the need for any customisations.

10.6 Future Research Opportunities

The following potential research areas are suggested to continue the evolution of the MDEIS framework and expansion of its capabilities and application.

10.6.1 Production MDEIS Build

During the conduct of this thesis research and investigations, a comprehensive model design has been developed and is available in full as per the Appendix. The CASE designer is capable of generating code templates and database schemas, the latter which was often used to simulate and test some of the model logic segments during design and refinement.

A full implementation of a production quality MDEIS framework including the main components of the MDEIS editor with runtime engines suitable to allow user interaction on a commonly available platform will demonstrate the expected useability and efficiency of the solution, and provide an ongoing basis for iterative refinement of the model and framework components.

A key element of the user access needs to be aimed at and involve business users acting as logic definers to promote the accessibility of the framework by non-technical users.

10.6.2 Additional Distributed Instance Integrations

During the development of this thesis a later addition to the model was the Distributed Instance Integration where I defined a simple model of inter-MDEIS-

instance integration between similar MDEIS model instances. In addition to the authorisation structures I identified four specific integrations denoted Distributed Execution Requests, namely, Data Replication, Key Authorization, Logic Variant and Workflow Triggers.

Once a core MDEIS framework solution has been developed, these distributed extensions are relatively simple additions to be implemented. Further, they provide opportunity for further investigation:

- **Additional DERs**: that can provide different integration functionality between the similar model instances.
- **Generic Integration Options**: that might be able to be defined between any distributed model instances, not just where they utilise similar models. Any such generic integration options are more complex as they involve the integration of potentially completely different streams, requiring definition of the disparate integration points as well as identifying the alternate logic paths and ensuring mutual compatibility, an issue that is otherwise simplified with similar models. Such distributed model integrations would be analogous to integrating any logic segments between any other applications.

The latter investigations may also need to be considered in conjunction with the defined Application Model Merging options that I have already defined for directly integrating discrete application models operating on the same instance.

10.6.3 Additional Logic Extensions

This thesis has been directed to modelling solutions for EIS style applications due to their relative structural simplicity and repetition of information management structures. There is no fundamental reason why additional functionality could not be defined that would readily enable the MDEIS framework to extend to other application domains.

All that is required to extend the framework is; definition of the new model segments and logic, encapsulation of the new elements into the runtime engines and model editor.

Some manipulation of the core model structure may be required to fully cater for a fully plug'n'play style capability for such third party extensions.

10.6.4 Permit Temporal Update

A key capability of the combined temporal management of the application meta-data model and the application data is that the MDEIS framework supports full temporal execution across the entire lifecycle of the application. Nominally, you would expect that any temporal execution sessions would only be permitted to operate in a read-only manner, preserving the temporal integrity of the application.

It is nevertheless possible to remove the read-only restriction from a temporal execution session. The runtime engine can thus also optionally permit permanent temporal data changes to be made to the end database rather than restricting operation to a read only mode. I refer to such changes as a permanent temporal update however as this is implementing a historic data change then there is an extra level of due diligence required by the authorised user in exercising this option, in addition to the inherent data integrity limitations that need to be imposed by the runtime engine.

While the framework's temporal management is capable of managing any such changes to data existing prior to the selected temporal execution date, including adding new data, any preferred effects on data that was subsequently defined (at a later temporal date) cannot always be readily interpreted. For the temporal editor, this will require potential further manual data interpretation and intervention.

To aid the temporal editor, this will require the development of to-be-defined additional temporal management capabilities to identify any subsequently important or affected data (that was temporally defined at a later period) according to to-be-defined relevant association rules that the temporal editor might be interested in. Further, associated actions such as targeted record deletion or other logical actions to then be performed on such identified records through the temporal stream would also need to be developed for action by the temporal editor.

Obviously, providing such additional capabilities to alter temporal data present the potential for some risk to the database, although only due to its potential as any batch data change operation might have. The nature of the temporal data management will protect the fundamental data integrity in any case whilst implemented.

10.6.5 Runtime Security Monitoring

The MDEIS framework supports a comprehensive security access model to limit user access to data and model objects using a variety of group and individual user means. However, there may be occasions in secure operating environments when

certain legitimate user access may be required to trigger additional monitoring or oversight operations.

As the framework supports the definition of workflows on any model object, these workflow triggers can be utilised to initiate any further required actions. As the framework only provides the definition of generic workflow options, which might become cumbersome to manage in a high definition volume environment, it is likely that additional monitoring management functionality could readily be defined as a model extension.

10.6.6 Visual Semantic Debugging

While any model editor must work within the core model syntax and internal object and relationship rules, even the best model-based IDE cannot guarantee that every idea that a user or logic definer has will be able to be successfully translated into a working set of model objects.

Indeed, while the correct model syntax may be guaranteed by the model editor, users still need to analyse whether they have semantically achieved a correct outcome. While the runtime environment should provide sandboxed simulations to the logic definers, and automated documentation can be generated that reflect the operation of the modelled objects, complicated logic (in particular) may still be semantically incorrect.

As the model repository maintains the full logic definition of the modelled objects, it can also track every event, data change and workflow step that occurs in complete fine detail. Instead of relying on traditional forms of debugging lists this atomic level of object and action granularity offers the potential for advanced visual debugging displays that can graphically display event trees and data transactions using novel node-based style graphics to aid in more clearly understanding all triggered events and workflows throughout every execution step.

10.7 Conclusion

Worldwide commercial, product driven and custom built software represent massive global development efforts. Unfortunately, much of this effort also involves major levels of duplication, repetition and re-work as well as the additional efforts to test, verify and deploy software applications. Regrettably, much of the software

industry still resembles the early generation industries where many of the developed products are often one of a kind and inflexible. Further, a large proportion of larger scale EIS style application developments end in failure. Apart from poor project management and development practices, failures can also often be attributed to software complexity, rapid requirements evolution, diversity or expansion in platforms, user bases and collaborations requirements.

Progressive improvements in methodologies, toolsets and technology platforms continue to provide scope for incremental optimisation and efficiency improvements but often at their own significant costs in continual re-training and re-development. Model based alternatives have long been provided as a subset of these improvements, in mostly niche or layer based solutions, however they typically only provide a partial optimisation service. There is a desperately needed paradigm shift required in software development that can truly reduce the scale of technological barriers and increase the openness of what many customers experience as a closed or locked in application environment.

Expanding a model based approach to software application development whereby all aspects of the logical application requirements are captured in a meta-data model from which the ultimate software application is directly executed from – with no direct programming – is such a paradigm shift. Greatly increasing the global effectiveness of such a paradigm shift is the removal of the main technological development barriers - the need for highly trained technical software programmers - instead utilising existing business analysts, power users and even normal business users to define their requirements into the model for direct application execution.

The associated potential decrease in the ongoing operational costs for organisations is significant. The additional productivity potential is even greater by empowering our ever increasing technologically savvy workforce to become logic definers to improve the efficiency of their own workflows and processes.

Appendices

Thesis Attachments

The following additional documents are provided on supplementary media as they represent significant details and works generated throughout the research resulting in this thesis.

Full Distributed Temporal Meta-Data EIS Application Model

The design for the meta-data EIS application model was modelled using Sybase PowerDesigner, primarily using combinations of the design software's Conceptual Data and UML Class Diagram modelling features to capture the core details of the meta-data EIS application model.

The design elements are primarily defined in platform inspecific dimensions, however the models can be readily transformed into any of the supported platform specific physical implementation models as required for ongoing development of the runtime environments.

Throughout the theses only summary design excerpts have been extracted to provide visual representations of each of the primary model components in as simple a format and structure as reasonably possible.

The PowerDesigner toolset does provide the ability to extract a full and complete object listing in a structured report format with the full details all of the modelled objects, their relationships and descriptions. This includes more detailed model descriptions and options than are explicitly described in this thesis. This generated document is somewhat larger than this thesis document, in the order of 2000 pages itself, so has not been included as a printed component of this thesis, and is included as an attachment.

Published Papers

The full text of all refereed papers (see full list on page 5) that were published by the author during the research are included as attachments.

Glossary

Acronym	Expansion
ADL	Architecture Description Language
API	Application Programming Interface
ARIS	Architecture of Integrated Information Systems
BA	Business Analysts or Analysis
BASIC	Beginner's All-purpose Symbolic Instruction Code
BPDM	Business Process Definition Metamodel
BPEL	Business Process Execution Language
BPML	Business Process Modelling Language
BPMN	Business Process Model and Notation
C2	Chiron-2
CASE	Computer Aided Software Engineering
CIM	Computer Independent Model
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
COTS	Commercial Off The Shelf
CRUD	Create, Read or Retrieve, Update and Delete
CSS	Cascading Style Sheets
CWM	Common Warehouse Metamodel
DBMS	Database Management System
DC	Distributed Components
DEIS	Distributed Enterprise Information System
DER	Distribution Execution Requests
DIS	Diagram Interchange Specification
DR	Data Replication
DSL	Domain-Specific Language
DSM	Domain-Specific Modelling
DSML	Domain-Specific Modelling Language
DSDM	Dynamic Systems Development Method

Acronym	Expansion
EA	Enterprise Architecture Enterprise Architect
EE	Element Envelopment
EIS	Enterprise Information System
EPC	Event-driven Process Chain
EPM	Eclipse Modelling Project
ERP	Enterprise Resource Planning
GUI	Graphical User Interface
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IS	Information Systems
ISO	International Organisation for Standardisation
IT	Information Technology
JBPM	Java Process Definition Language
JIT	Just-In-Time
KA	Key Authorization
LV	Logic Variant
MBDM	Model Based Development Methodology
MDA	Model Driven Architecture
MDD	Model Driven Design
MDE	Model Driven Engineering
MDEIS	Meta-Data EIS Application
MIS	Management Information Systems
MOF	MetaObject Facility
MOF2T	MOF to Text
MVC	Model-View-Controller
NASA	National Aeronautics and Space Administration
OASIS	Organization for the Advancement of Structured Information Standards
OEM	Original Equipment Manufacturer
OMG	Object Management Group

Acronym	Expansion
OO	Object-Oriented
OODBMS	Object-Oriented DBMS
ORDBMS	Object-Relational DBMS
ORM	Object Relational Mapper
PHP	PHP: Hypertext Preprocessor
PIM	Platform Independent Model
PSM	Platform Specific Model
QoS	Quality of Service
QVT	Query-View-Transformation
RAD	Rapid Application Development
RDBMS	Relational Database Management System
RMI	Remote Method Invocation
RUP	Rational Unified Process
SER	Standard Element Referencing
SME	Subject Matter Experts Small to Medium Sized Enterprise
SOA	Service-Oriented Architecture
SQL	Structured Query Language
UI	User Interface
UML	Unified Modelling Language
VDEM	Virtual Data Element Mapping
VL	Variant Logic
XAML	Extensible Application Mark-up Language
XMI	XML Metadata Interchange
XML	Extensible Mark-up Language
XP	Extreme Programming
XPDL	XML Process Definition Language
XSD	XML Schema Definition
XUL	XML User Interface Language
W3C	World Wide Web Consortium
WfMC	Workflow Management Coalition

Acronym	Expansion
WS-CDL	Web Services Choreography Description Language
WWW	World Wide Web

References

1. Coplien, J. *Objects of the people, by the people, and for the people*. in *11th International Conference on Aspect-oriented Software Development*. 2012. Potsdam, Germany.
2. The Standish Group International. *Chaos, Project Smart*. 2014 [cited 2014 20 Sept 2014]; Available from: http://www.projectsmart.co.uk/docs/chaos-report.pdf.
3. Com, Q.B., *Quotes about Time and Time Management*. 2012: Quotations Book.
4. Mayer, J.J., *If You Haven't Got the Time to Do It Right, When Will You Find the Time to Do It Over?* 1990: Simon and Schuster.
5. expertglossary.com, *Enterprise Information System - Definition*, in *Java Platform, Enterprise Edition (Java EE) 1.4 Glossary*. 2013.
6. Tsadimas, A., et al. *Evaluating software architecture in a model-based approach for enterprise information system design*. in *Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge*. 2010. Cape Town, South Africa, ACM: 72-79.
7. Davis, J. and E. Chang, *Variant Logic for Model Driven Applications*, in *Advances and Applications in Model-Driven Software Engineering*, D.V.G. Díaz , et al., Editors. 2013, IGI Global. p. 1-34.
8. Gilb, T. *Evolutionary project management: Multiple performance, quality and cost metrics for early and continuous stakeholder value delivery*. in *International Conference on Enterprise Information Systems*. 2004. Porto.
9. Sauer, C. and Cuthbertson, *The State of IT Project Management in the UK 2002-2003.* Computer Weekly Project/Programme Management Survey, 2003.
10. The Standish Group International. *2004 Third Quarter Research Report: Chaos Demographics*. 2004; Available from: http://standishgroup.com/sample_research/PDFpages/q3-spotlight.pdf.
11. Coplien, J. *Organizational patterns: Beyond technology to people*. in *International Conference on Enterprise Information Systems*. 2004. Porto.
12. Benington, H.D., *Production of Large Computer Programs.* IEEE Annals of the History of Computing, 1983. **5**(4): p. 350–361.

13. Sommerville, I., *Software Engineering*. 6th Edition ed. 2000: Addison-Wesley Publishing Company.
14. Birrell, N.D. and M.A. Ould, *A practical handbook to software development*. 1988: Cambridge University Press.
15. SDLC. *Spiral Model*. Software Development Life Cycle 2011 [cited 2013 18/4/2013]; Available from: http://www.sdlc.ws/spiral-model/.
16. Forsberg, K. and H. Mooz. *The relationship of system engineering to the project cycle*. in *NCOSE*. 1991. Chattanooga, Tennessee.
17. OMG. *Unified Modelling Language*. 2003; Available from: http://www.uml.org/.
18. Hoda, R., et al. *Agility in context*. SIGPLAN Not. 45(10): 74-88.
19. Version One. *Agile Methodologies for Software Development*. 2013 [cited 2013 18/4/2013]; Available from: http://www.versionone.com/Agile101/Agile-Development-Methodologies-Scrum-Kanban-Lean-XP/.
20. DSDM Consortium, *DSDM Atern V2*. 2008: DSDM Consortium.
21. APMG. *Agile Project Management™ Certification*. 2013 [cited 2013; Available from: http://www.apmg-international.com/en/qualifications/agile-pm/agile-pm.aspx.
22. King, J. and L. Williams. *Log your CRUD: design principles for software logging mechanisms* in *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security*. 2014. Raleigh, North Carolina, USA, ACM: 1-10.
23. dictionary.com. *Rapid Application Development definition*. dictionary.com 2013; Available from: http://dictionary.reference.com/browse/rapid+application+development.
24. Rouse, M., *CASE (computer-aided software engineering)*, in *SearchCIO-Midmarket*. 2005, TechTarget.
25. Davies, I.G.a.G., Peter, F. and Rosemann, Michael and Gallo, Stan. *Conceptual Modelling – What and Why in Current Practice*. in *23rd International Conference on Conceptual Modelling (ER'04)*. 2004. Shanghai, China: Springer.
26. Jiang, Z. and P. Naudé, *An Examination of the Factors Influencing Software Development Effort*. International Journal of Computer and Information Engineering, 2007. **1**(3).

27. Hitchins, D.K., *Advanced Systems Thinking, Engineering, and Management*. 2012: Artech House.
28. Alves, C. and A. Finkelsteiin, *Challenges in COTS decision-making: a goal-driven requirements engineering perspective*, in *Proceedings of the 14th international conference on Software engineering and knowledge engineering*. 2002, ACM: Ischia, Italy. p. 789-794.
29. Megas, K., et al., *A study of COTS integration projects: product characteristics, organization, and life cycle models*, in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. 2013, ACM: Coimbra, Portugal. p. 1025-1030.
30. Beatty, R.C. and C.D. Williams, *ERP II: best practices for successfully implementing an ERP upgrade.* Commun. ACM, 2006. **49**(3): p. 105-109.
31. Wailgum, T., *Mission Impossible: Judging TCO of Enterprise Software Upgrades*, in *CIO*. 2014, CXO Media Inc.
32. Laguna, M. A. and B. Gonzalez-Baixauli. *Requirements variability models: meta-model based transformations* in *Proceedings of the 2005 symposia on Metainformatics*. 2005. Esbjerg, Denmark, ACM: 9.
33. Davis, J. and E. Chang. *Lifecycle and generational application of automated updates to MDA based enterprise information systems*. in *International Symposium on Information and Communication Technology*. 2011. Hanoi, Viet Nam: Association for Computing Machinery.
34. Davis, J., A. Tierney, and E. Chang. *Meta Data Framework for Enterprise Information Systems Specification - Aiming to Reduce or Remove the Development Phase for EIS Systems*. in *6th International Conference Enterprise Information Systems*. 2004. Porto, Portugal: ICEIS 2004.
35. Chang, E., J. Davis, and S.K. Chalup. *A New Look At the Enterprise Information System Life Cycle - Introducing the Concept of Generational Change*. in *International Conference on Enterprise Information Systems*. 2003.
36. ISO. *We're ISO, the International Organization for Standardization. We develop and publish International Standards.* 2014 [cited 2014 15-Dec-2014]; Available from: http://www.iso.org/iso/home.html.
37. Office of Government Commerce, *Managing Successful Projects with PRINCE2*. 2009, UK: The Stationary Office.

38. PMI, *A Guide to the Project Management Body of Knowledge*. 2013: Project Management Institute.
39. Royce, W.W., *Managing the development of large software systems.* IEEE Wescon, 1970(August): p. 1-9.
40. Ruparelia, N.B., *Software development lifecycle models.* SIGSOFT Softw. Eng. Notes, 2010. **35**(3): p. 8-13.
41. K. Forsberg and H. Mooz. *The Relationship of Systems Engineering to the Project Cycle*, in *Proceedings of the First Annual Symposium of the national Council on Systems Engineering (NCOSE)*. 1991. Chattanooga, Tennessee.
42. Mooz, H. and K. Forsberg, *A visual explanation of the development methods and strategies including the waterfall, spiral, vee, vee+, and vee++ models*. 2001. p. 4-6.
43. I., J., B. G., and R. J., *The unified software development process*. 1999, Reading, Massachusetts: Addison-Wesley.
44. Mostafa, A., et al. *Toward a Formalisation of UML2.0 Metamodel using Z Specifications*. in *8th International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*. 2007. IEEE Computer Society Washington, DC, USA.
45. Beck, K. and et al. *Manifesto for Agile Software Development*. 2001; Available from: http://www.agilemanifesto.org.
46. Maurer, F. and G. Melnik, *Agile methods: moving towards the mainstream of the software industry*, in *Proceedings of the 28th international conference on Software engineering*. 2006, ACM: Shanghai, China. p. 1057-1058.
47. Gotterbarn, D., *UML and agile methods: in support of irresponsible development.* SIGCSE Bull., 2004. **36**(2): p. 11-13.
48. Raltus. *Agile / Scrum*. IT QA and Development Methodologies 2013 [cited 2013 18/4/2013]; Available from: http://www.raltus.com/SolutionsITQADevelopment.htm#SDLCScrum.
49. Elshamy, A. and A. Elssamadisy, *Applying agile to large projects: new agile software development practices for large projects*, in *Proceedings of the 8th international conference on Agile processes in software engineering and extreme programming*. 2007, Springer-Verlag: Como, Italy. p. 46-53.
50. Martin, J., *Rapid Application Development*. 1991, Indianapolis, IN, USA: Macmillan Publishing Co., Inc.

51. Malone, K. and J. Griffith. *A case study in the use of Groovy and Grails* in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. 2012, Trento, Italy, ACM: 1254-1255.
52. Agarwal, R., et al., *Risks of rapid application development.* Commun. ACM, 2000. **43**(11es): p. 1.
53. Howard, A., *Rapid Application Development: rough and dirty or value-for-money engineering?* Commun. ACM, 2002. **45**(10): p. 27--29.
54. Muller, M.M. and F. Padberg, *On the economic evaluation of XP projects.* SIGSOFT Softw. Eng. Notes, 2003. **28**(5): p. 168-177.
55. Nosek, J., *The case for collaborative programming.* Communications of the ACM, 1998. **41**(3): p. 105-108.
56. Cockburn, A. and L. Williams., *The costs and benefits of pair programming*, in *eXtreme Programming and Flexible Processes in Software Engineering XP2000*. 2000: Cagliari, Italy.
57. Kuppuswami, S., et al., *The effects of individual XP practices on software development effort.* SIGSOFT Softw. Eng. Notes, 2003. **28**(6): p. 6-6.
58. APMG. *APM Group and DSDM Consortium join forces to deliver innovative Agile Project Management Certification*. 2010 [cited 2013; Available from: http://www.apmgroupltd.com/PressCentre/20Sep2010AgilePMCert.asp.
59. Pressman, R., *Software Engineering: A Practitioner's Approach*. 5th ed. 2001: McGraw-Hill Inc.
60. Martin, R.H. and D. Raffo, *A model of the software development process using both continuous and discrete models.* Software Process: Improvement and Practice, 2000. **5**(2-3).
61. Donzelli, P. and G. Iazeolla, *A hybrid software process simulation model.* Software Process: Improvement and Practice, 2001. **6**(2).
62. Abrahamsson, P., et al. *New directions on agile methods: a comparative analysis*. 2003. Portland, Oregon: IEEE Computer Society.
63. Jiang, L. and A. Eberlein, *Towards a framework for understanding the relationships between classical software engineering and agile methodologies*, in *Proceedings of the 2008 international workshop on Scrutinizing agile practices or shoot-out at the agile corral*. 2008, ACM: Leipzig, Germany. p. 9-14.

64. Avison, D.E. and G. Fitzgerald, *Where now for development methodologies?* Commun. ACM, 2003. **46**(1): p. 78--82.
65. Veryard, R. *Component-Based Development*. 1999 [cited 2013; Available from: http://www.users.globalnet.co.uk/~rxv/CBDmain/cbdfaq.htm.
66. Frakes, W.B. and C.J. Fox, *Sixteen questions about software reuse.* Commun. ACM, 1995. **38**(6): p. 75--ff.
67. Karhinen, A., A. Ran, and T. Tallgren. *Configuring designs for reuse*. 1997. Boston, Massachusetts, United States: ACM Press.
68. Holmes, R. and R.J. Walker, *Systematizing pragmatic software reuse.* ACM Trans. Softw. Eng. Methodol., 2013. **21**(4): p. 1-44.
69. Henninger, S., *An evolutionary approach to constructing effective software reuse repositories.* ACM Trans. Softw. Eng. Methodol., 1997. **6**(2): p. 111--140.
70. Senthil, R., et al. *An improved component model for component based software engineering*. 2007. SIGSOFT Softw. Eng. Notes 32(4): 9.
71. Cheong, Y.C. and S. Jarzabek. *Frame-based method for customizing generic software architectures*. 1999. Los Angeles, California, United States: ACM Press.
72. Ravichandran, T., *Special issue on component-based software development.* SIGMIS Database, 2003. **34**(4): p. 45-46.
73. Lim, W.C., *Strategy-driven reuse: Bringing reuse from the engineering department to the executive boardroom.* Annals of Software Engineering, 1998(5): p. 85-103.
74. Due, R., *The economics of componentbased development.* Information Systems Management, 2000. **17**(1): p. 92-95.
75. Provensi, L. L., et al. *Self-adaptive middleware for digital ink based applications* in *Proceedings of the 7th workshop on Reflective and adaptive middleware*. 2008. Leuven, Belgium, ACM: 29-34.
76. Ye, C., et al. *Middleware support for internetware: a service perspective* in *Proceedings of the Second Asia-Pacific Symposium on Internetware*. 2010. Suzhou, China, ACM: 1-10.
77. OMG. *OMG Corba*. 2004; Available from: http://www.corba.org/.
78. Microsoft. *Microsoft COM*. 2002; Available from: http://www.microsoft.com/com/.

79. Oracle. *Remote Method Invocation Home*. 2013 [cited 2013; Available from: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html.
80. Medvidovic, N., *On the role of middleware in architecture-based software development*, in *Proceedings of the 14th international conference on Software engineering and knowledge engineering*. 2002, ACM: Ischia, Italy. p. 299-306.
81. Colman, A., et al., *Adaptive application-specific middleware*, in *Proceedings of the 1st workshop on Middleware for Service Oriented Computing (MW4SOC 2006)*. 2006, ACM: Melbourne, Australia. p. 6-11.
82. Blair, G.S., et al., *The role of ontologies in emergent middleware: supporting interoperability in complex distributed systems*, in *Proceedings of the 12th International Middleware Conference*. 2011, International Federation for Information Processing: Lisbon, Portugal. p. 400-419.
83. Costa, M. B., et al. *Software frameworks for information systems integration based on web services* in *Proceedings of the 2008 ACM symposium on Applied computing*. 2008. Fortaleza, Ceara, Brazil, ACM: 777-782.
84. Hamu, D.S. and M.E. Fayad, *Achieving bottom-line improvements with enterprise frameworks.* Commun. ACM, 1998. **41**(8): p. 110--113.
85. Madsen, K. *Five years of framework building: lessons learned*. 2003. Anaheim, CA, USA: ACM Press.
86. Fayad, M.E. and D.S. Hamu, *Enterprise frameworks: guidelines for selection.* ACM Comput. Surv., 2000. **32**(1es): p. 4.
87. Microsoft. *Three-Layered Services Application*. 2003 [cited 2013; Available from: http://msdn.microsoft.com/en-us/library/ff648105.aspx.
88. Steiert, H.-P., *Towards a component-based n-Tier C/S-architecture*, in *Proceedings of the third international workshop on Software architecture*. 1998, ACM: Orlando, Florida, USA. p. 137-140.
89. Cemus, K., et al. *Enterprise information systems: comparison of aspect-driven and MVC-like Approaches* in *Proceedings of the 2015 Conference on research in adaptive and convergent systems*. 2015. Prague, Czech Republic, ACM: 330-336.
90. Anttonen, M., et al. *Transforming the web into a real application platform: new technologies, emerging trends and missing pieces* in *Proceedings of the*

2011 ACM Symposium on Applied Computing. 2011. TaiChung, Taiwan, ACM: 800-807.

91. Aghaee, S. and C. Pautasso, *Mashup development with HTML5*, in *Proceedings of the 3rd and 4th International Workshop on Web APIs and Services Mashups*. 2010, ACM: Ayia Napa, Cyprus. p. 1-8.
92. Microsoft. *Microsoft Windows Code-Named "Longhorn" Development Centre*. 2004 [cited 2004 May 15th]; Available from: http://msdn.microsoft.com/longhorn/.
93. MyXAML. *Welcome to MyXaml!* 2004 [cited 2004; Available from: http://www.myxaml.com/.
94. Xamlon. *xamlon: accelarating development*. 2004 [cited 2004; Available from: http://www.xamlon.com/.
95. Mozilla. *XML User Interface Language (XUL)*. 2004 [cited 2004 April 20th]; Available from: http://www.mozilla.org/projects/xul/.
96. Mozilla. *Firefox*. 2013 [cited 2013; Available from: http://www.mozilla.org/en-US/firefox/new/.
97. Wigmore, I. *Definition: business logic*. 2013 [cited 2013; Available from: http://whatis.techtarget.com/definition/business-logic.
98. Curbera, F., et al., *The next step in Web services.* Commun. ACM, 2003. **46**(10): p. 29-34.
99. Jordan, D., et al., *Web services business process execution language version 2.0.* OASIS Standard, 2007. **11**.
100. ISO. *Information Technology – Database Languages – SQL – Part 1: Framework*. 2003; Available from: http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=34132.
101. Devarakonda, R. S. *Object-relational database systems - the road ahead*. 2001. Crossroads 7(3): 15-18.
102. Bancilhon, F. *Object databases*. 1996. ACM Comput. Surv. 28(1): 137-140.
103. Hibernate. *Object to Relational Mapping and Relationships with Hibernate*. 2003; Available from: http://www.hibernate.org/.
104. Pohjalainen, P. and J. Taina, *Self-configuring object-to-relational mapping queries*, in *Proceedings of the 6th international symposium on Principles and practice of programming in Java*. 2008, ACM: Modena, Italy. p. 53-59.

105. Bayrak, C. and C. Davis, *The liquid architecture: a non-linear peer-to-peer distributed architecture with polymorphic message passing.* SIGSOFT Softw. Eng. Notes, 2003. **28**(3): p. 2-2.
106. Microsoft. *Microsoft Small Basic*. 2013 [cited 2013; Available from: http://www.microsoft.com/en-us/download/details.aspx?id=22961.
107. Oracle. *Java EE at a Glance*. 2013 [cited 2013; Available from: http://www.oracle.com/technetwork/java/javaee/overview/index.html.
108. Microsoft. *Visual Studio Ultimate 2012*. 2013 [cited 2013; Available from: http://www.microsoft.com/visualstudio/eng/products/visual-studio-ultimate-2012#product-edition-ultimate.
109. Microsoft. *Visual Studio Team Foundation Server*. 2013; Available from: http://www.microsoft.com/visualstudio/eng/products/visual-studio-team-foundation-server-2012#product-edition-tfs.
110. dictionary.com. *standard*. 2013; Available from: http://dictionary.reference.com/browse/standard.
111. OMG. *Business Process Model and Notation (BPMN)*. 2011 [cited 2013; Available from: http://www.omg.org/spec/BPMN/index.htm.
112. Perry, S. *When is a Process Model Not a Process Model - A Comparison Between UML and BPMN*. in *Process Modelling Using UML, 2006. The IEE Seminar on (Ref. No. 2006/11432)*. 2006.
113. OASIS. *OASIS Web Services Business Process Execution Language (WSBPEL) TC*. 2013 [cited 2013 2013]; Available from: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.
114. Hojsgaard, E. and T. Hallwyl. *Core BPEL: syntactic simplification of WS-BPEL 2.0* in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. 2012. Trento, Italy, ACM: 1984-1991.
115. Hallwyl, T., F. Henglein, and T. Hildebrandt, *A standard-driven implementaion of WS-BPEL 2.0*, in *Proceedings of the 2010 ACM Symposium on Applied Computing*. 2010, ACM: Sierre, Switzerland. p. 2472-2476.
116. Workflow Management Coalition. *Workflow Management Coalition*. 2013 [cited 2013 2013]; Available from: http://www.wfmc.org/.
117. Guelfi, N. and A. Mammar. *A formal framework to generate XPDL specifications from UML activity diagrams* in *Proceedings of the 2006 ACM symposium on Applied computing*. 2006. Dijon, France, ACM: 1224-1231.

118. Genchev, O. and J. Galletly, *XPDL: bringing business and software together - a case study*, in *Proceedings of the International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*. 2009, ACM: Ruse, Bulgaria. p. 1-6.
119. Mladenova, M. and B. Zhelyazova. *Competitive analysis of software solutions for business organizations* in *Proceedings of the 2007 international conference on Computer systems and technologies*. 2007. Bulgaria, ACM: 1-8.
120. Chen, L., Q. Jiayin, and S. Huaying. *A SOA-Based ARIS Model for BPR*. in *e-Business Engineering, 2008. ICEBE '08. IEEE International Conference on*. 2008.
121. Hajmoosaei, M., et al. *Towards a change-aware process environment for system and software process* in *Proceedings of the 2015 International Conference on Software and System Process*. 2015. Tallinn, Estonia, ACM: 32-41.
122. Berenbach, B. A. *Comparison of UML and text based requirements engineering* in *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*. 2004. Vancouver, BC, CANADA, ACM: 247-252.
123. Medvidovic, N., et al., *Modeling software architectures in the Unified Modeling Language.* ACM Trans. Softw. Eng. Methodol., 2002. **11**(1): p. 2-57.
124. Pardillo, J., *A systematic review on the definition of UML profiles*, in *Proceedings of the 13th international conference on Model driven engineering languages and systems: Part I*. 2010, Springer-Verlag: Oslo, Norway. p. 407-422.
125. Vincent, M., *Communicating requirements for business: UML or problem frames?*, in *Proceedings of the 3rd international workshop on Applications and advances of problem frames*. 2008, ACM: Leipzig, Germany. p. 16-22.
126. Schmidt, D., *Introduction: Model-Driven Engineering.* IEEE Computer Science, 2006. **39**(2): p. 25-31.
127. Ortiz, G. and B. Bordbar. *Aspect-Oriented Quality of Service for Web Services: A Model-Driven Approach*. in *IEEE International Conference on Web Services*. 2009. IEEE Computer Society Washington, DC, USA.

128. Cicchetti, A., D. Di Ruscio, and A. Di Salle. *Software customization in model driven development of web applications*. in *2007 ACM symposium on Applied computing*. 2007. ACM, New York, NY, USA.
129. Fabra, J., et al. *Enabling the Evolution of Service-Oriented Solutions Using an UML2 Profile and a Reference Petri Nets Execution Platform*. in *3rd International Conference on Internet and Web Applications and Services*. 2008. IEEE Computer Society Washington, DC, USA.
130. Zhu, X. and S. Wang. *Software Customization Based on Model-Driven Architecture Over SaaS Platforms*. in *International Conference on Management and Service Science*. 2009. CORD Conference Proceedings.
131. France, R. and B. Rumpe. *Model-driven Development of Complex Software: A Research Roadmap*. in *Future of Software Engineering*. 2007. IEEE Computer Society Washington, DC, USA.
132. Motik, B., A. Maedche, and R. Volz, *A Conceptual Modeling Approach for Semantics-Driven Enterprise Applications*, in *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, R. Meersman and Z. Tari, Editors. 2002, Springer Berlin Heidelberg. p. 1082-1099.
133. OMG. *About the Object Management Group™ (OMG™)*. 2004 [cited 2003 August 20th]; Available from: http://www.omg.org/gettingstarted/gettingstartedindex.htm.
134. OMG. *OMG Model Driven Architecture. The Architecture of Choice for a Changing World*. 2012 [cited 2012 May 13]; Available from: http://www.omg.org/mda/.
135. The Eclipse Foundation. *Eclipse Modeling Project*. 2013 [cited 2013; Available from: http://www.eclipse.org/modeling/.
136. OMG. *Search for Member Companies*. 2013 [cited 2013; Available from: http://www.omg.org/cgi-bin/apps/membersearch.pl.
137. Kruchten and Philippe, *The Rational Unified Process: An Introduction*. 2000: Addison-Wesley Pub Co.
138. Soden, M., H. Eichler, and J. Hoessler. *Inside MDA: Mapping MOF2.0 Models to Components*. 2003; Available from: http://modeldrivenarchitecture.esi.es/pdf/paper1-3.pdf.
139. OMG. *MDA Guide V1.0.1*. 2012 [cited 2013 2013]; Available from: http://www.omg.org/cgi-bin/doc?omg/03-06-01.

140. IBM. *Rational Rose*. 2004; Available from: http://www-306.ibm.com/software/rational/.
141. Quatrani, T. *Introduction to the Unified Modelling Language*. in *Rational User Conference*. 2001.
142. Armano, G. and M. Marchesi, *A rapid development process with UML*. SIGAPP Appl. Comput. Rev., 2000. **8**(1): p. 4--11.
143. Lethbridge, T. and R. Laganiere, *Object-Oriented Software Engineering: Practical Software Development using UML and Java*. 2002: McGraw-Hill.
144. Allegrini and Tiziana, *Code Generation Starting From Statecharts Specified in UML*. 2001: Universita Degli Studi di Pisa.
145. Guizzardi, G., H. Herre, and G. Wagner, *Towards ontological foundations for UML conceptual models*, in *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*. 2002, Springer. p. 1100-1117.
146. Kendall, E.F. and M.E. Dutra, *An Introduction and UML Profile for the Web Ontology Language (OWL)*. 2002: Sandpiper Software Inc.
147. uml-diagrams.org. *UML 2.4 Diagrams Overview*. 2013 [cited 2013 2013]; Available from: http://www.uml-diagrams.org/uml-24-diagrams.html.
148. Jackson, E. K., et al. *Components, platforms and possibilities: towards generic automation for MDA* in *Proceedings of the tenth ACM international conference on Embedded software*. 2010. Scottsdale, Arizona, USA, ACM: 39-48.
149. H., O.J., *Architecture for Software Intensive Products and Systems – An International Perspective*. 2000: Phillips Research.
150. The Eclipse Foundation. *About the Eclipse Foundation*. 2013 [cited 2013; Available from: http://www.eclipse.org/org/.
151. The Eclipse Foundation. *All Eclipse Foundation Members*. 2013 [cited 2013; Available from: http://www.eclipse.org/membership/showAllMembers.php.
152. The Eclipse Foundation. *Eclipse Modeling Project Charter*. 2013 [cited 2013; Available from: http://www.eclipse.org/modeling/modeling-charter.php.
153. The Eclipse Foundation. *List of Projects*. 2013 [cited 2013; Available from: http://projects.eclipse.org/list-of-projects.
154. The Eclipse Foundation. *Eclipse Project Status*. 2013 [cited 2013; Available from: http://www.eclipse.org/projects/tools/status.php?top=modeling.

155. Luftman, J., R. Papp, and T. Brier, *Enablers and inhibitors of business-IT alignment.* Commun. AIS, 1999. **1**(3es): p. 1.
156. Butler, K.A. *Designing DEEPER: towards a user-centered development environment*. 1995. Ann Arbor, Michigan, United States: ACM Press.
157. Rajkovic, P., et al. *Software Tools for rapid Development and Customization of Medical Information Systems*. in *12th IEEE International Conference on e-Health Networking Applications and Services*. 2010. IEEE Computer Society Washington, DC, USA.
158. Hagen, C. and G. Brouwers. *Reducing Software Life-Cycle Costs by Developing Configurable Software*. in *Aerospace and Electronics Conference*. 1994. IEEE Press Washington, DC, USA.
159. Nitu. *Configurability in SaaS (software as a service) applications*. in *2nd India software engineering conference*. 2009. ACM, New York, NY, USA.
160. De Alwis, B. and J. Sillito. *Why are software projects moving from centralized to decentralized version control systems?* in *2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*. 2009. IEEE Computer Society Washington, DC, USA.
161. Ren, Y., et al. *Software Configuration Management of Version Control Study Based on Baseline*. in *3rd International Conference on Information Management, Innovation Management and Industrial Engineering*. 2010. IEEE Press Washington, DC, USA.
162. Kaur, P. and H. Singh, *Version Management and Composition of Software Components in Different Phases of the Software Development Life Cycle.* ACM Sigsoft Software Engineering Notes, 2009. **34**(4): p. 1-9.
163. Koegel, M., et al. *Operation-based conflict detection*. in *1st International Workshop on Model Comparison in Practice*. 2010. ACM, New York, NY, USA.
164. Steinholtz, B. and K. Walden, *Automatic Identification of Software System Differences.* IEEE Transactions on Software Engineering, 1987. **SE-13**(4): p. 493-497.
165. Ma, X., et al. *Version-consistent dynamic reconfiguration of component-based distributed systems*. in *19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 2011. ACM, New York, NY, USA.

166. Jansen, S., S. Brinkkemper, and R. Helms. *Benchmarking the Customer Configuration Updating Practices of Product Software Vendors*. in *7th International Conference on Compostion Based Software Systems*. 2008. IEEE Computer Society Washington, DC, USA.
167. Brown, A., *Oops! Coping With Human Error in IT.* ACM Queue – System Failures, 2004. **2**(8): p. 34-41.
168. Press., O.U., *model*, in *Oxford University Press*. 2014, Oxford University Press.
169. Schnabel, F., et al., *Empowering Business Users to Model and Execute Business Processes*, in *Business Process Management Workshops*, M. Muehlen and J. Su, Editors. 2011, Springer Berlin Heidelberg. p. 433-448.
170. Barry & Associates Inc. *Service-Oriented Architecture (SOA) Definition*. 2104 [cited 2014; Available from: http://www.service-architecture.com/articles/web-services/service-oriented_architecture_soa_definition.html.
171. Chanda, J., et al. *Behavioral and structural evolution of SOA from OO: an integrated approach*. 2013. SIGSOFT Softw. Eng. Notes 38(5): 1-9.
172. Garlan, D., *Software architecture: a roadmap*, in *Proceedings of the Conference on The Future of Software Engineering*. 2000, ACM: Limerick, Ireland. p. 91-101.
173. Chen, Y. and R. Sion. *To cloud or not to cloud?: musings on costs and viability* in *Proceedings of the 2nd ACM Symposium on Cloud Computing*. 2011, Cascais, Portugal, ACM: 1-7.
174. Bailey, K.D., *Methods of Social Research*. 1999: Diane Publishing Company.
175. Blalock, A.B. and H.M. Blalock, *Introduction to social research*. 1982: Prentice-Hall.
176. Nunamaker, J.F., M. Chen, and T.D. Purdin, *Systems Development in Information Systems Research.* Journal of Management Information Systems, 1991. **7**(3): p. 89-106.
177. Mctavish, D.G. and H.J. Loether, *Social Research: An Evolving Process*. 2002, Boston: Allyn and Bacon, Longmans, Inc.
178. Trochim, W.M. *Research Methods Knowledge Base*. 2002 [cited 2014; Available from: http://www.anatomyfacts.com/research/researchmethodsknowledgebase.pdf.

179. Burstein, F. and S. Gregor. *The Systems Development or Engineering Approach to Research in Information Systems: An Action Research Perspective*. in *10th Australasian Conference in Information Systems*. 1999. Wellington, New Zealand,.
180. Kaplan, B. and J. Maxwell, *Qualitative Research Methods for Evaluating Computer Information Systems*. Evaluating Health Care Information Systems: Methods & Applications, ed. J. Anderson, C. Aydin, and S. Jay. 1994, Thousand Oaks, California: Sage.
181. Galliers, R.D., *Information Systems Research: Issues, Methods and Practical Guidelines*. 2nd ed. 2002, London: Palgrave.
182. Denning, P.J. and et al, *Computing as a Discipline.* Communications of the ACM, 1968. **11**(5): p. 323-333.
183. Rouse, M. *enterprise architecture (EA)*. 2014 [cited 2014; Available from: http://searchcio.techtarget.com/definition/enterprise-architecture.
184. Gartner. *Gartner IT Glossary > Enterprise Architecture (EA)*. 2013 [cited 2014; Available from: http://www.gartner.com/it-glossary/enterprise-architecture-ea/.
185. Cameron, B. H. *Enterprise systems education: new directions & challenges for the future* in *Proceedings of the 2008 ACM SIGMIS CPR conference on Computer personnel doctoral consortium and research*. 2008. Charlottesville, VA, USA, ACM: 119-126.
186. Hendrick, S. and K. Hendrick. *PowerDesigner 10.0's Expanding Role in Model - Driven Development*. 2004 [cited 2004 15th December]; Available from: http://www.sybase.com/content/1033448/IDCwhitepaper.pdf.
187. W3C. *XForms - The Next Generation of Web Forms*. 2004 [cited 2004 May 17th]; Available from: http://www.w3.org/MarkUp/Forms/.
188. Grimes, R., *Developing Applications with Visual Studio .NET*. 2002: Addison-Wesley Pub Co.
189. Celms, E., A. Kalnins, and L. Lace. *Diagram definition facilities based on metamodel mappings* in *18th International Conference, OOPSLA'2003 (Workshop on Domain-Specific Modeling*. 2003.
190. Davis, J. and E. Chang. *Variant Logic Meta-data Management for Model Driven Applications - Allows Unlimited End User Configuration and Customisation of All Meta-data EIS Application Features*. in *International*

Conference on Enterprise Information Systems 2011. Beijing, China: SciTePress.

191. Apple. *iOS Developer Library* 2014 [cited 2014; Available from: https://developer.apple.com/library/ios/navigation/.
192. Google. *Package Index*. 2014 [cited 2014; Available from: http://developer.android.com/reference/packages.html.
193. Microsoft. *API reference for Windows Runtime apps*. 2014; Available from: http://msdn.microsoft.com/en-au/library/windows/apps/br211369.aspx.
194. Ecma International. *ECMAScript Programming Language*. 2014 [cited 2014; Available from: http://www.ecmascript.org/.
195. The jQuery Foundation. *jQuery - Category: Ajax*. 2014 [cited 2014; Available from: http://api.jquery.com/category/ajax/.
196. Adobe. *Flex Documentation*. 2014 [cited 2014; Available from: http://www.adobe.com/devnet/flex/documentation.html.
197. Microsoft. *About the Silverlight Documentation*. 2014 [cited 2014; Available from: http://msdn.microsoft.com/en-us/library/cc189047(v=vs.95).aspx.
198. W3C. *HTML5 - A vocabulary and associated APIs for HTML and XHTML*. 2014 [cited 2014; Available from: http://www.w3.org/TR/html5/.
199. Gartner. *Gartner Says Worldwide Traditional PC, Tablet, Ultramobile and Mobile Phone Shipments to Grow 4.2 Percent in 2014*. 2014 [cited 2014; Available from: http://www.gartner.com/newsroom/id/2791017.

* * *