# Merging Application Models in a MDA Based Runtime Environment for Enterprise Information Systems

Jon Davis[1], Andrew Tierney[2] and Elizabeth Chang[3]

[1]J. Davis, Curtin University of Technology, Bentley, 6102, Australia, Jon.Davis@cbs.curtin.edu.au
[2]A. Tierney, Curtin University of Technology, Bentley, 6102, Australia, Andrew.Tierney@cbs.curtin.edu.au
[3]E. Chang, Curtin University of Technology, Bentley, 6102, Australia, Elizabeth.Chang@cbs.curtin.edu.au

*Abstract*—**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 relatively recent field of Model Driven Architecture is primely involved in the definition and development of the source model structures for model based applications and in developing transformations from the abstract models to various executable formats. The authors are also involved in these MDA activities in the development of their G2 prototype system targeted at the specific domain of Enterprise Information System style applications. They have reviewed various methods for merging application models within this domain and describe the fundamentals of three application model integration methods from their G2 system; Standard Element Referencing, Virtual Data Element Mapping and Element Envelopment that can be used to provide practical and simple application model merging at both the design time and runtime of a model based production system, to produce a working integrated merged application.**

*Index Terms*—**Model Driven Architecture, application models, model management, merging models, Enterprise Information Systems.**

## I. Introduction

The overwhelming majority of computer applications in existence today are the result of highly labour intensive and repetitive development activities. Traditional system development methodologies such as the Waterfall, Spiral, Fountain and V models [1] have not been fundamentally altered as a result of modern technologies and in general the software development industry still maintains variations of the basic paradigm for system development; analysis, design, develop code, test and deploy. New system development methodologies such as Prototyping, Agile Processes and the Big Ball of Mud [2], [3], [4] typically propose differing levels of task decompositions, parallelism and end-user interaction and can provide specific advantages when dutifully employed but they are not guaranteed to necessarily change the magnitude of the total effort.

Since 1997, the Object Management Group (OMG) has championed the introduction of Model Driven Architecture (MDA) [5] as its candidate solution to fundamentally improve on current development processes. MDA provides a guiding methodology based on the use of Unified Modelling Language (UML) [6] to define a Platform Independent Model (PIM) that represents an abstract yet accurate model of the requirements of an application, following through with the efficient development and deployment of the final target system as an instance of the then Platform Specific Model (PSM).

There is an increasingly large body of research that is investigating various aspects of MDA related work including; PIM model definition, automating the transformation process from the PIM to the PSM, and further separation of the application layers such as the user interface layers to facilitate deployment of MDA applications to any platform.

MDA has not yet been realised as a proven concept let alone as a deployable technology and there is a healthy degree of scepticism as to whether it will prove to become worthwhile. Accordingly the bulk of MDA research is motivated to developing the basic technology alternatives that will establish MDA as a realistic and practical system development option.

The authors of this paper are proponents of MDA and are constructing a new version of their model based system called G2 which is aimed at the domain of Enterprise Information System (EIS) style applications [7].

In developing G2, we have also assumed that the success of MDA is assured (at some stage) and have further considered development activities in a post MDA-technology era where we believe that the current focus on developmental and technological issues such as components, reuse and cross platform compatibility will become obsolete and replaced by the concepts of Model Development and Model Integration (MI) as the primary focus. We justify this statement by the acknowledgement that when and if MDA succeeds, the PIM model of applications effectively becomes the only variable in application development (except for the ongoing development of the core runtime components that would support the execution of the PSM model). Once the PIM has been defined then the PSM transformations should automatically generate the application for any supported platform using the most current generation of the common runtime components that constitute the execution environment.

While we firmly believe that MDA will revolutionise the development of software and technically make all software available to all users, the field of Model Integration will offer the ability to mix and match model based applications and application fragments to form new model based applications that will more readily conform to the exacting requirements of each user and user group. MDA success also implies the potential availability of end-user model definition and editing capability for non-technical users (or at least not very technical users), which when combined with Model Integration will extend the capability of application model authoring, merging and editing to the common user and usher in a new age of accessible and efficient computer usage and information access.

In this paper we review the progress of application merging efforts, identify areas requiring further attention and present the Model Integration opportunities that we are including in our G2 system.

## II. Related Works

In traditional application development where the majority of the source is hand coded, the issue of merging applications is very problematic. This is largely due to the variability of application source code due to issues such as; the availability of the source code at all in some cases, the degree of the use of comments to document the code, the availability of any model files used to fully or partially generate the code, the availability of supporting system design documents, the programming language(s) and versions used to develop the system, the use of third party components, the age of the source code and currency of development tools, the relative nationality of the developers of the source code in terms of coding practices such as object naming, and the variety of platforms and application layers employed.

As is evident, the level of variability in some or all of these factors can very quickly escalate the issue of source code usability to the status of virtually unusable as has been well known throughout the software industry since the early days of software development. In many well documented cases, applications are rarely merged – rather than attempting to merge the applications the software developer will reproduce the functionality of the smaller application as a redevelopment of the larger application. [8]

When source code is not available, alternative integration methods such as; "screen scraping" to interoperate applications via automated manipulation of the user interface, direct manipulation of the application's database, and use of an application's Application Programming Interface (API) if available, are often used to access an application's features although these are application integration options rather than application merging options [9].

The introduction of newer technology such as; components, multi-tiered applications, CASE tools, UML and other modelling tools have all served to incrementally reduce overall development effort. Various other methods that utilise text processing and syntax analysis of the source code [10], [11] have attempted to assist in application merging and can produce syntactically correct merged application code however they do rely heavily on the use of common source code bases, which is not a typical occurrence by observation of the previously stated variability factors.

In the relatively recent field of Model Driven Architecture, majority effort is expended towards the definition and development of the models and the associated transformation of the abstract models to either direct execution or for source code generation that can then be modified and compiled to produce the required system [12], [13], [14]. There is a substantial volume of publications on generalised ontologies but minimal available literature that views past the current technical problems towards scenarios when MDA systems may become *de rigueur*.

## III. Model Integration Options

Application models and ontologies in general are commonly represented as directed graphs. With the existence of two or more models, say $M_1$, $M_2$, … etc, the combined set of vertices from the models becomes available to define additional new edges in the new combined model and thus provide integration between the models.

In a context free graph, model integration is extremely simple as an edge can be defined to connect any vertex to any other vertex. In a highly structured application model such as a PIM representation, each vertex must be typed to some degree in order to provide the correct context within the model syntax or structure. The practical restrictions of a domain specific application model (as used in G2) necessarily imposes limitations on which vertices can be logically joined by an edge. Note that to avoid confusion with object terminology we refer to model vertices as elements.

This paper does not address the direct editing of models and model elements as this process is identical (within each particular domain solution) regardless of whether there is one model or multiple models merged together into a combined but not yet integrated model. We do define these major classes of application model integration in the G2 system:

- **Standard Element Referencing** – the simplest merge option involves creating new references in one model to existing elements in a second model to provide access to application features of the second model to users of the first model.
- **Virtual Data Element Mapping** – provides infrastructure 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.

### *A. Standard Element Referencing (SER)*

When considering the functionality of the applications represented by the models, the simplest and therefore most likely initial model integration points would be expected to be:

- Merging application model menu elements to create the appearance of a single integrated application. i.e. the creation of a new and logical combined application menu access structure can immediately present the impression of a fully integrated suite of applications – the omission of unwanted, inapplicable or duplicated application fragments from the merged structure is also an important initial consideration when merging application models.

- Providing simple access to features of another application by creating new references or shortcuts to major functions of the secondary application model from within existing interfaces of the primary application model. e.g. defining new interface items such as buttons etc on existing forms of the primary application to invoke events or initiate other forms using functionality from the secondary application can provide an even higher level of apparent integration in the merged application model without yet modifying the underlying core logic of the modelled applications.

While the SER process is not strictly limited to Model Integration activities, as it is also a standard model editing feature, it does represent the simplest method to readily achieve basic model integration.

The management of SER 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 SER type modifications do not fundamentally alter the context of the models' individual workflows then they are classed as a safe model modification. Figure 1 indicates the application model structure changes as a result of SER type changes.

## *B. Virtual Data Element Mapping (VDEM)*

It is difficult to conceive of two 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 entity representation of one or more of these in some derivative form. Virtual Data Element Mapping is the process of identifying similar data elements in application models, and defining the basic rules that will merge the data elements at the underlying physical level.

While VDEM model integration may have no obvious impact in the application execution of the model as perceived by the end-user (which is actually one of its benefits) it does achieve a progressive integration of the underlying data by effectively merging relationally similar data and thus seamlessly providing access to the combined data for all of the originating applications.

Additionally the VDEM process does automate the cross application availability of the user interface and workflow support for all virtual attributes of the newly combined virtual entity which may then be observed by application users as new functionality in existing applications. i.e. the requirement to satisfy mandatory entries for virtual attributes that initially existed only in the other application model would automatically become additional features of the second and additional applications as managed by the model runtime engine.

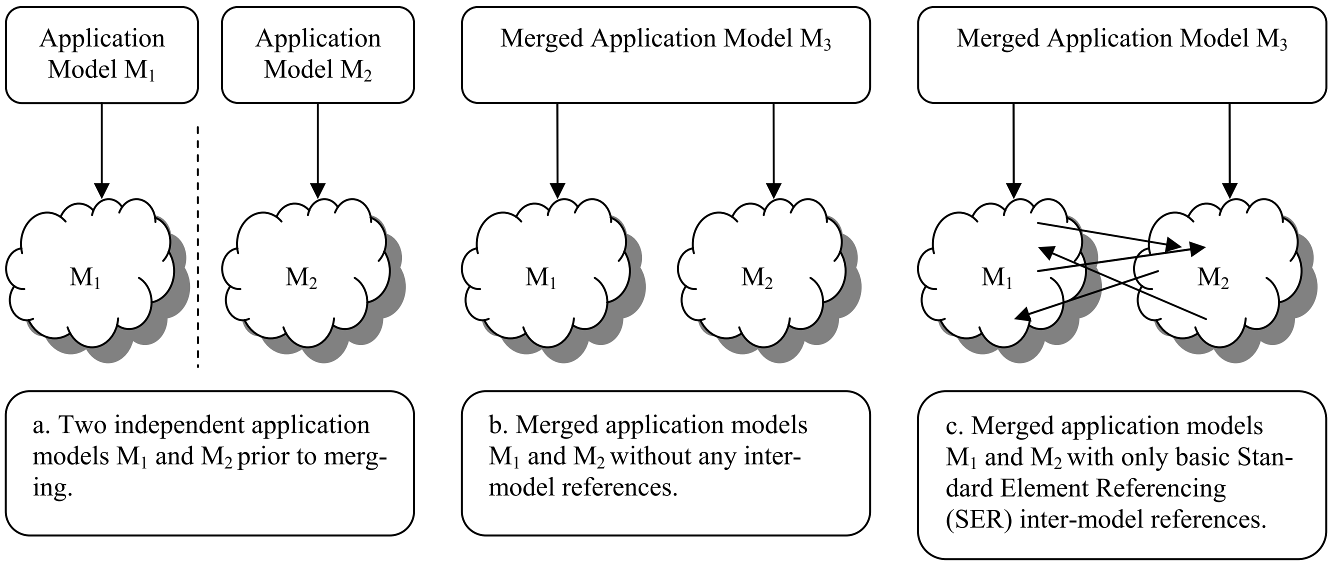

Application Model $M_1$
Application Model $M_2$
Merged Application Model $M_3$
Merged Application Model $M_3$
$M_1$
$M_2$
$M_1$
$M_2$
$M_1$
$M_2$
a. Two independent application models $M_1$ and $M_2$ prior to merging.
b. Merged application models $M_1$ and $M_2$ without any inter-model references.
c. Merged application models $M_1$ and $M_2$ with only basic Standard Element Referencing (SER) inter-model references.

**Fig. 1,** The initial steps in merging separate application models into a new model, creating basic inter-model integration between the models using only Standard Element Referencing (SER) inter-model references.

By progressively identifying and effecting the VDEM process for all similar entities between the merged application models, a fully integrated underlying database structure is created that services all merged applications without the ongoing inefficiencies that duplication of data causes. e.g. application model $M_1$ contains customer information in virtual entity JOB while model $M_2$ uses virtual entity JOBNUM (using a different virtual attribute definition) – by merging these two virtual entities with the VDEM process, the unioned set of associated virtual attributes becomes automatically available to both applications including the associated logic required to access, enter and update associated virtual attributes of the entire unioned set (see Figure 2 for an example of the VDEM process outcome).

The VDEM process consists of two steps. The first involves specifying the association rules for the identified virtual entities and their component virtual attributes to ensure valid interoperation of the merged virtual entity. A second optional process is required if there is any data in any of the candidate virtual entities, to effectively merge the data.

**Step 1. Virtual Entity Association (VEA)**: requires the identification and pairing of virtual entities from each application model that are very similar - in relational database terms the analogy is creating a 1:1 join between tables. Each virtual attribute either has a mapping relationship to one or more virtual attributes in the other virtual entity or it is considered independent. Independent virtual attributes do not require any further specification as all their existing rules are known and now consistent for all models – the remaining non-independent virtual attributes require only the definition of a mapping function for equivalence with the other virtual entity. All database constraint equivalent rules are preserved and unchanged. From the perspective of each original application model, virtual attributes from the other virtual entity that require mandatory access will automatically become available as new user interface features in each application – no changes are required to effect this although user interface aesthetics would be expected to trigger a manual forms edit at some stage rather than rely on the automated generation output from the model runtime engine.

**Step 2. Virtual Data Association (VDA)**: is only effected when physical data exists in at least one of the virtual entities. i.e. the application model merging is occurring at runtime in a production system rather than at design time. All existing data in both virtual entities is transformed

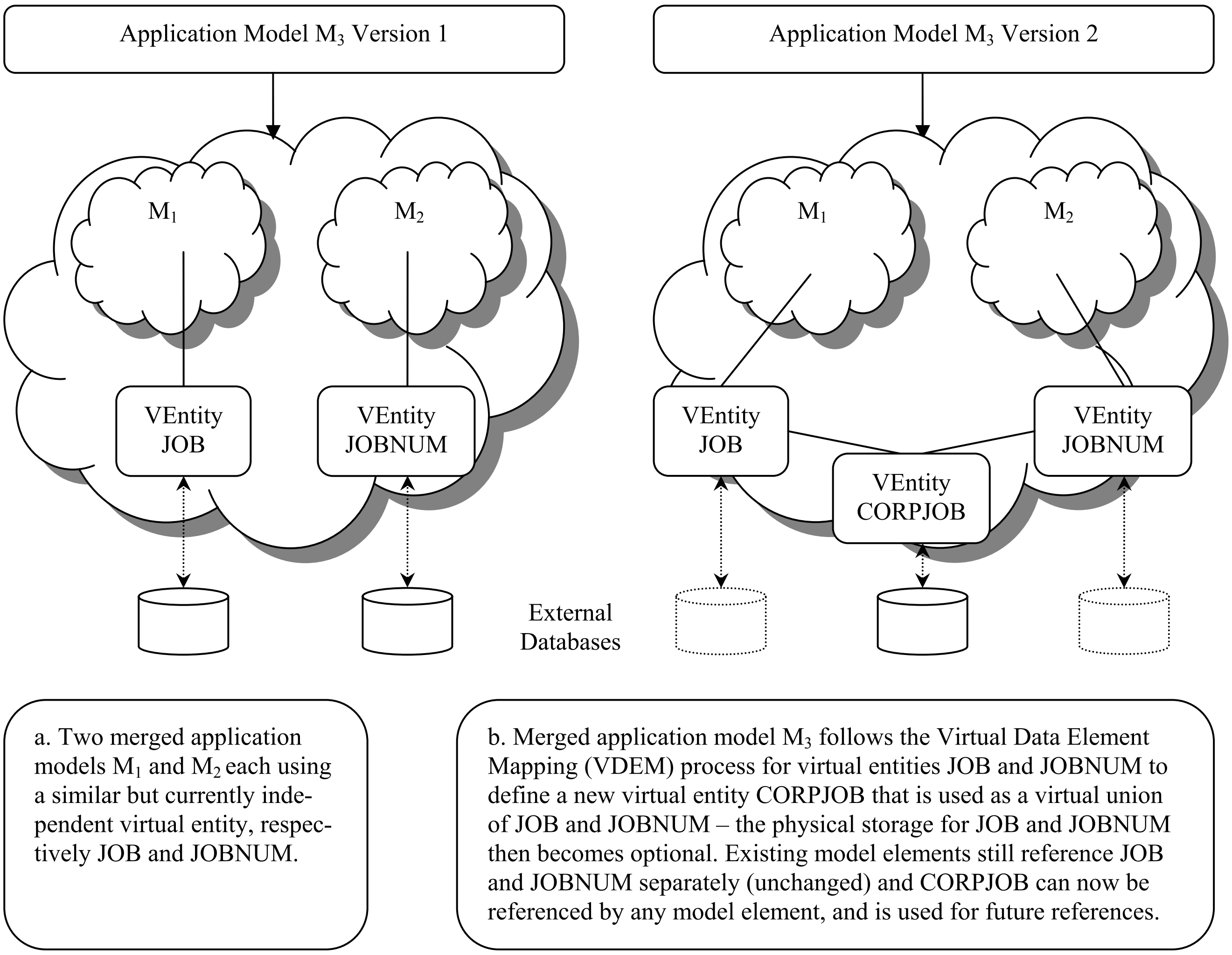

Application Model M3 Version 1
Application Model M3 Version 2
M1
M2
VEntity JOB
VEntity JOBNUM
VEntity CORPJOB
External Databases
a. Two merged application models M1 and M2 each using a similar but currently independent virtual entity, respectively JOB and JOBNUM.
b. Merged application model M3 follows the Virtual Data Element Mapping (VDEM) process for virtual entities JOB and JOBNUM to define a new virtual entity CORPJOB that is used as a virtual union of JOB and JOBNUM – the physical storage for JOB and JOBNUM then becomes optional. Existing model elements still reference JOB and JOBNUM separately (unchanged) and CORPJOB can now be referenced by any model element, and is used for future references.

**Fig. 2,** Following application model merging, data level integration processes utilising the Virtual Data Element Mapping (VDEM) process facilitates data merging at the application model and physical levels.

based on the mapping functions defined during VEA, including duplicate detection and new data migration. The level of automation of VDA is dependent on the selection of user intervention options and the level of duplication in the data.

The outcomes of the VDEM process are as follows:

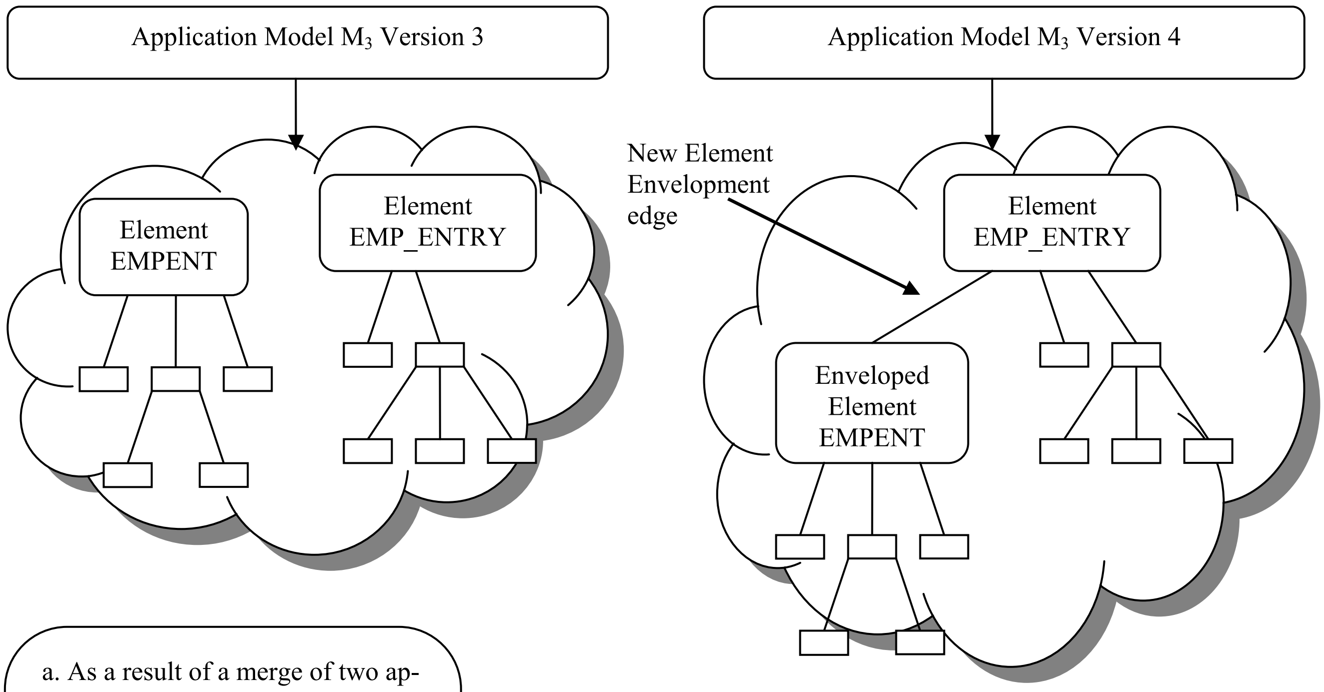
- Each original sub-model is unchanged and still references the original virtual entity and virtual attributes.
- A new virtual entity is created as a merge of the two other virtual entities, which is automatically referenced by the model runtime engine, and is available for future referencing within the model.
- Any data in the original two virtual entities is merged into the new virtual entity. Continuing physical update into the original storage for the original virtual entities becomes optional.
- Additional application model user interface and workflow elements are automatically defined (from standard patterns) where any new mandatory virtual attributes from the new virtual entity have been created with respect to each merged application sub-model.

A likely third step in the VDEM process is to "walk the graph" of the meta-model and fully replace all references to both of the original virtual entities with the corresponding references to the new virtual entity, thus resulting in a fully integrated model specification for the benefit of future editing. This is a simple model editor toolset function that reassigns the model's virtual entity and attribute references accordingly.

The management of VDEM type model integration options requires familiarity with the data structures of the modelled applications although not at the technical database level. Performing the VDEM process is not analogous to advanced database management as the merged model already contains and abstracts all database details from the VDEM user. Some knowledge of the available system functions is required in order to create mapping functions as required.

### *C. Element Envelopment (EE)*

As useful as class inheritance is in the object world, so is the inheritance of elements in ontologies such as application models. Model elements that have repeated use within an application but within different contexts (often with minimal or no change) use element inheritance to speed and simplify model editing.


Application Model $M_3$ Version 3
Element EMPENT
Element EMP_ENTRY
Application Model $M_3$ Version 4
New Element Envelopment edge
Element EMP_ENTRY
Enveloped Element EMPENT

a. As a result of a merge of two application models, the merged application has two very similar elements which are currently independent, respectively EMPENT and EMP_ENTRY and their inherited elements. e.g. user interface fields for the entry of employee identifier from the two original applications.

b. Merged application model $M_3$ follows the Element Envelopment (EE) process for elements EMPENT and EMP_ENTRY by enveloping EMPENT as a virtual inheritance of EMP_ENTRY. The new Element Envelopment edge may require additional transformation metadata to be defined to ensure any typing compatibility. Future changes to element EMP_ENTRY will now be inherited by elements that previously inherited from EMPENT.

**Fig. 3,** Following application model merging, element rationalisation utilising the Element Envelopment (EE) process facilitates merging user interface and workflow elements at the application model level.

When multiple models are merged, in an analogy to the VDEM processes for similar data level elements, other merged model elements may perform similar functions and should be rationalised into a cohesive 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.

Element Envelopment is the process of identifying functionally similar elements in application models, and merging the element to be enveloped (and any descendent elements of that element) into the element inheritance structure of the other identified element. The EE process is fairly simple in that it merely requires the identification of the root nodes for each of the elements that are to be subjected to the EE process. The root node of the element to be enveloped is then enveloped as a new descendent into the other element's element inheritance structure at a selected level (see Figure 3 for an example).

In the case where the enveloped element is of a different element type or there is a difference in the i/o patterns of the element, then a mapping function may need to be specified between the enveloped element and its new antecedent to satisfy model integrity. This only needs to occur initially and between these two elements as standard element inheritance is satisfied for all other elements in the joined inheritance structure. To complete the envelopment process requires the determination of which standard element attributes are inherited by the enveloped element which is a standard inheritance feature for each edge.

The management of EE type model integration requires higher familiarity with the overall model structures of the modelled applications, the meta-model attributes, and of the available system functions is required in order to create mapping functions.

## IV. Recapitulation, Conclusions and Future Works

The merging of models is necessarily domain specific. This paper has presented the practical integration options for the G2 system, an instantiation of the domain of Enterprise Information System style application models.

The integration methods presented are:

- Standard Element Referencing
- Virtual Data Element Mapping
- Element Envelopment

Other forms of model integration such as merging multiple versions of a single application are more readily available by using standard graph processing and comparison algorithms as have been well researched in the public domain for other ontological domains [15], [16]. Such comparative algorithms that expect a high degree of correlation in order to automate our process of Element Envelopment within the models is a fairly simple extension.

## V. References

[1] I. Sommerville, *Software Engineering (6th Edition)*, Addison-Wesley Pub Co, 2000.

[2] R. Pressman, *Software Engineering: A Practitioner's Approach 5th Ed*, McGraw-Hill Inc, 2001.

[3] R.H. Martin and D. Raffo, "A model of the software development process using both continuous and discrete models", *Proceedings of Software Process: Improvement and Practice*, Volume 5, Number 2-3, June-September 2000.

[4] P. Donzelli, G. Iazeolla, "A hybrid software process simulation model", *Proceedings of Software Process: Improvement and Practice*, Volume 6, Number 2, June 2001.

[5] "OMG Model Driven Architecture – The Architecture of Choice for a Changing World", *http://www.omg.org/mda/*, 2003.

[6] "Unified Modelling Language", *http://www.uml.org/*, 2003.

[7] J. Davis, A. Tierney, E. Chang, "Meta Data Framework for Enterprise Information Systems Specification - Aiming to Reduce or Remove the Development Phase for EIS Systems", in *Proceedings of the 6th International Conference Enterprise Information Systems*, 2004, pp. 451-456.

[8] W. Emmerich, "Distributed component technologies and their software engineering implications", in *Proceedings of the 24th International Conference on Software Engineering*, 2002, pp. 537-546.

[9] M.B. Juric, I. Rozman, M. Hericko and T. Domajnko, "Integrating legacy systems in distributed object architecture", *SIGSOFT Softw. Eng. Notes* 25, 2000, pp 35-39.

[10] V. Berzins, "Software merge: semantics of combining changes to programs" ,*ACM Trans. Program. Lang. Syst.* 16, 1994, pp 1875-1903.

[11] S. Horwitz, J. Prins and T. Reps, "Integrating noninterfering versions of programs", *ACM Trans. Program. Lang. Syst.* 11, 1989, pp 345-387.

[12] A. Tolk, "Avoiding Another Green Elephant – A Proposal for the Next Generation HLA Based on the Model Driven Architecture", in *Fall Simulation Interoperability Workshop*, 2002.

[13] S. Melnik, E. Rahmand P.A. Bernstein, "Rondo: a programming platform for generic model management", in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, 2003, pp. 193-204.

[14] S.J. Mellor, "Make models be assets", in *Commun. ACM 45*, 2000, pp 76-78.

[15] Luqi, "A Graph Model for Software Evolution", in *IEEE Trans. Softw. Eng.* 16, 1990. pp 917-927.

[16] S. Uchitel and M. Chechik, "Merging partial behavioural models", in *Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering*, 2004, pp 43-52.

[17] M. Gruninger and J. Lee, "Ontology applications and design: Introduction", in *Commun. ACM* 45, 2002, pp 39-41.