Working Paper Series
ISSN 1170-487X

Providing Integrated Support for Multiple Development Notations

by John C. Grundy and John R. Venable

Working Paper 94/17
November, 1994

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

Providing Integrated Support for Multiple Development Notations

John C. Grundy and John R. Venable

Abstract

A new method for providing integrated support for multiple development notations (including analysis, design, and implementation) within Information Systems Engineering Environments (ISEEs) is described. This method supports both static integration of multiple notations and the implementation of dynamic support for them within an integrated ISEE. First, conceptual data models of different analysis and design notations are identified and modelled, which are then merged into an integrated conceptual data model. Second, mappings are derived from the integrated conceptual data model, which translate data changes in one notation to appropriate data changes in the other notations. Third, individual ISEEs for each notation are developed. Finally, the individual ISEEs are integrated via an integrated data dictionary based on the integrated conceptual data model and mappings. An environment supporting integrated tools for Object-Oriented Analysis and Extended Entity-Relationship diagrams is described, which has been built using this technique.

1. Introduction

1.1. Integrated ISEEs

A software system can be modelled using a variety of notations, such as Object-Oriented Analysis and Design (OOA/D) diagrams, Extended Entity-Relationship (EER) diagrams and Data Flow Diagrams (DFDs). The choice of modelling notation is often dependent on the kind of problem and organisational and designer preferences. Some problems suit being modelled using object-oriented techniques while others are more easily conceptualised using entity-relationship and data flow diagrams. The use of different notations for different (or the same) parts of a problem offers several advantages: the most appropriate notation can be used for each part; the same design can be expressed using different notations; different designers can communicate about the same design using different notations; and organisations using different notations can collaborate on projects. Integrated ISEE support for using different notations on the same project is necessary, however, to provide consistency management between each notation, and thus produce a consistent design for the problem as a whole. In fact, integrated ISEEs enable the use of multiple notations. Without them, effective use of multiple notations would not be feasible.

1.2. Our Approach

Providing integrated support for multiple development notations (whether to support single or multiple phases of development), requires both static and dynamic integration. We define static integration as the conceptual integration of the description languages (or notations) that are used by the system developers. This defines the requirements for how concepts in one notation map onto another notation. Dynamic integration is concerned with how changes in one system description (using whatever notation) are propogated to all other relevant system descriptions, whether in the same or different notations. Dynamic integration benefits extensively from computer-based tool support.

In this paper, we present a new approach which supports both the static and dynamic integration of multiple notations. Figure 1 illustrates this technique. The first 4 steps illustrate the steps in achieving static integration: 1) The conceptual data models of different design notations are identified and documented. 2) Commonalities between aspects of these notations' conceptual data models are identified, resulting in partial intgerated data models. 3) Mappings between different notation data model components are identified i.e. how changes to data in one model can be translated into changes to data in the other. 4) Several levels of partial data models can be built up, and these are collected to describe an integrated conceptual data model encompasing all of the design notations.

Authors' address: Department of Computer Science, University of Waikato, Private Bag 3105, Hamilton, New Zealand. e-mail: jgrundy@cs.waikato.ac.nz and jvenable@cs.waikato.ac.nz.

The second 4 steps define a dynamic integration of design and implementation tools supporting the different notations. The conceptual data models and mappings form the basis for implementing an integrated ISEE encorporating all of the design notations. 5) Individual tools for each notation are developed with the tool data dictionary based on the notation's conceptual data model. 6) Editors for this data dictionary are developed which may include multiple textual and graphical views for the notation. All of these tools have a consistent user interface provided by common building blocks used to implement the tools. 7) The integrated data model (and possibly the intermediate partial data models) are used to define an integrated data dictionary. 8) Changes to data in one notation's data dictionary are propagated to other notation data dictionaries via the integrated data dictionary. This translation of data changes is implemented as specified by the mappings defined in step 3 between the conceptual data models.

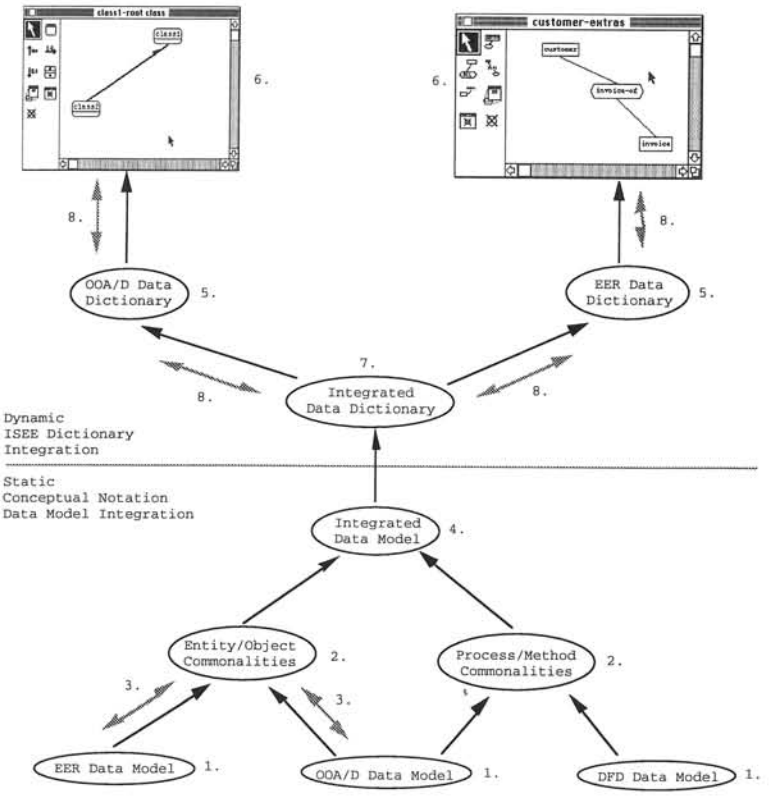

figure 1. Using an integrated, hierarchical data model to integrate different design notations.

In the following sections, related research in integrating design notations is discussed, and then an environment which incorporates integrated OOA/D and EER tools, called OOEER, is illustrated. The development of conceptual data models for the individual notations and an integrated conceptual data model is briefly described, together with mappings between the notations. Two previously developed OOA/D and EER environments are introduced and the implementation of OOEER using these environments is decribed. The paper concludes with a brief discussion of experience with this integrated tool and future research directions.

2. Related Research

Integrated ISEEs (or Integrated CASE tools and programming environments) allow users to analyse, design, and implement Information Systems from within one environment, providing a consistent user interace and consistent data dictionary (or repository). They help to minimise inconsistencies that can arise when using several separate tools for information systems development [14, 10].

The research in integrating ISEEs can be characterized as concentrating on how to achieve dynamic integration, via the implementation of models in integrated CASE (ICASE) or integrated programming environments. Some work has been done on the static integration of notations. Data modelling has been used to compare different notations [8] and support methodology engineering [7]. Process-modelling (which often includes some data modelling) has also been applied to comare and integrate notations [11]. However, there has been little work at developing detailed integration of individual notations. In particular,

little has been done to provide methods to translate conceptually integrated notations into tool-based implementations.

Limited dynamic notation integration is supported by many CASE tools, such as Softare Thru Pictures [14] and TurboCASE [12]. These ICASE environments allow developers to analyse and design software using a variety of different notations, with limited inter-notation consistency management. For example, entities from an EER model can be used as objects in an OOA model, with name and attribute changes propagated between each model. Such tools do not generally provide more complex mappings between the design notations, such as propagating an EER relationship addition to corresponding OOA diagram. This greatly limits their usefulness for supporting integrated development using different notations. The implementation of these environments is generally not sufficient to allow different design notations to be effectively integrated. These environments are generally separate from implementation tools, so consistency between design and implementation is not maintained.

FIELD environments [10] utilise selective broadcasting to keep muliple tools conistent under change. This involves propagating messages between separate Unix tools to maintain consistency between different views of software development. FIELD can not effectively integrate different design notation tools, however, as it provides no integrated data dictionary, and as more tools are added, any translation process becomes very complex and difficult to implement. Dora environments [9] provide abstractions for keeping multiple textual and graphical views consistent under change. These multiple views share the same data repository and hence can more easily be kept consistent. Dora does not, however, provide any mechanism for propagating changes between views which can not be directly applied by the environment. Some changes made to a design notation which can not be automatically implemented in another notation by the environment thus can not be supported.

Two key methods are required for dynamic integration: (1) utilising an integrated data dictionary/repository to automatically keep aspects of different notations' data consistent and (2) providing additional environment support for ensuring consistency between aspects which cannot be automatically kept consistent by the environment. Such aspects that require human intervention to maintain consistency should be indicated by the environment to the notation/tool user. This relies on being able to identify aspects that can be automatically kept consistent and those which require human intervention. Doing so requires effective static integration by identifying commonalities between the notation's conceptual data models, including change mappings between these data models.

3. A User's Perspective of OOEER

We have built an ISEE, called OOEER, which integrates the OOA/D and EER design notations, in addition to supporting object-oriented program implementation and relational schema definition. Unlike most CASE tools, OOEER propagates all changes made to OOA/D diagrams to related EER diagrams, and vice-versa. This not only includes simple mappings, such as entity, object and attribute addition, renaming and deletion, but also all relationship manipulation in both models. While the environment can only partially infer required changes to some diagrams when other notation diagrams are modified, it always informs designers of such changes made, to assist them in maintaining inter-notation consistency.

A screen dump from OOEER is shown in figure 2. This shows an OOA view (diagram) and an EER view for an invoicing Information System design. The dialog shows the changes that have been made to the customer OOA class. Items marked with a '*' were made to the customer entity in the EER view and translated by OOEER into changes on the customer class.

The OOA/D notation used is based on an early version of the MOSES notation [6]. Bold, arrowed lines represent generalisation relationships, thin lines represent aggregation and shaded lines represent association. Client/supplier relationships are represented by shaded lines with caller/called method names and (optional) method argument types. The EER notation used is based on a form of Chen's ER model [1]. Square icons represent entities with optional named attributes connected to the entity icon. Diamond-shaped icons represent relationships with optional role names and arities on the connecting lines. Subtyping is exclusive ('-' relationship) or inclusive ('+' relationship).

All of these views are kept consistent by OOEER. If a designer changes information in an OOA view, this change is propagated to all other views which share the changed information (other OOA, OOD, EER and implementation views). Where the environment can automatically make an appropriate change to keep these affected views consistent, the change is performed. For example, if the customer class were renamed in the

OOA view, the customer entity, method class name, and relational table would also be renamed. The reverse is also true if any of the other views with this information are updated.

For some view edits, OOEER can not directly update other affected views. For example, when a relationship is added between two entities in an EER view, the OOA view requires a relationship to be added. The EER update does not specify whether this relationship should be an association or aggregation relationship. OOEER by default adds an association relationhip between the two classes in the OOA view, but colours this relationship, indicating the designer needs to add further information. The designer can also view a description of the EER view change(s) made which affect the OOA view and which could not be fully implemented by the environment. The dialog in figure 2 shows some example EER changes on the customer entity and its relationships. Some changes made in the EER view require the user to further modify the OOA view, to ensure it is correct.

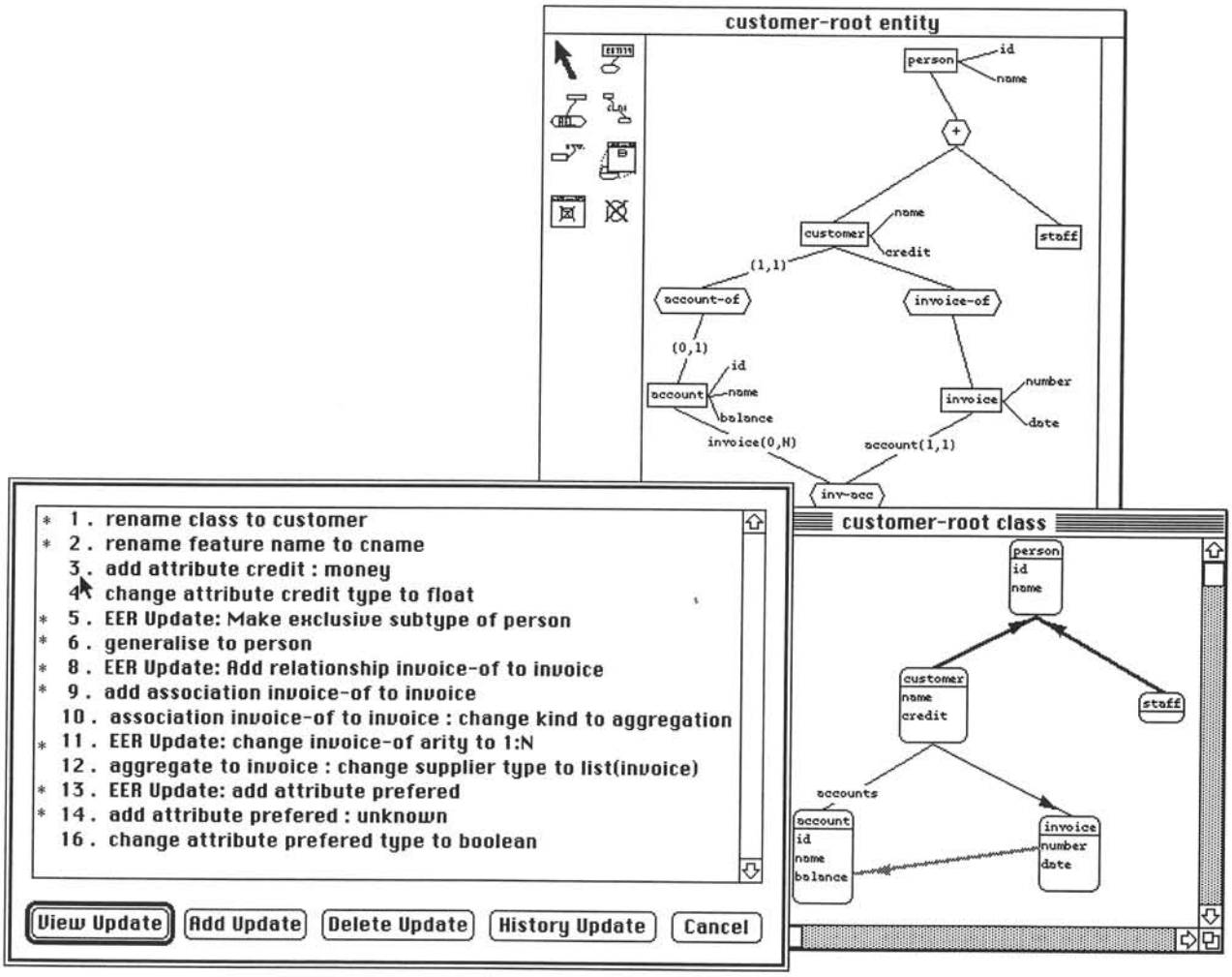

figure 2. A screen dump from the integrated OOEER environment.

OOEER also supports editable, textual object-oriented program views and relational schema views. These are kept consistent with the graphical views by expanding the change descriptions at the start of the textual view. These inform programmers of changes that need to be made to keep the implementation view consistent, as shown in figure 2. Designers can select change descriptions and request the environment try to automatically update the view, or can manually implement an appropriate change and then delete the change description. Thus unlike most CASE tools, OOEER supports integrated design and implementation views.

4. Conceptual Data Model Integration

The first 4 steps in the tool integration process for OOEER involve developing conceptual data models for the OOA/D and EER notations, and mappings between them. We have developed these conceptual models,

and a hiearchical, integrated conceptual model, using a conceptual data modelling language called CoCoA [13]. Figure 3 shows the conceptual data models for the EER and OOA/D notations used by OOEER.

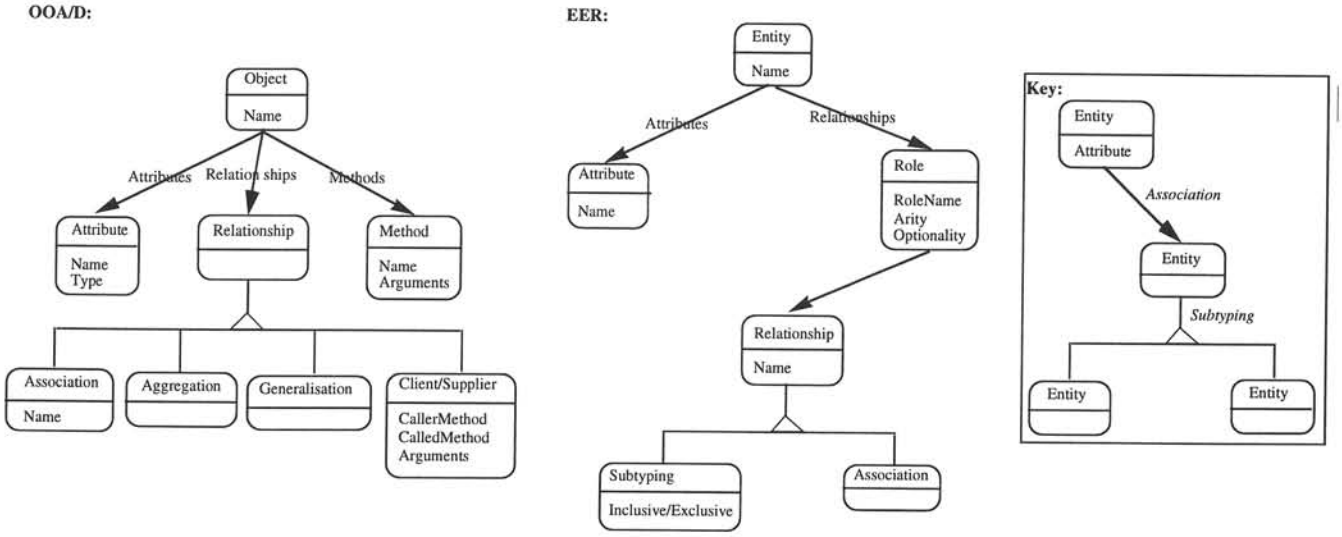

figure 3. The separate conceptual data models for OOA/D and EER notations.

The OOA/D notation defines named objects which have a collection of named, typed attributes and a collection of named methods with arguments. Generalisation relationships indicate an object is generalised to one (or possibly several) more abstract objects. Aggregation relationships indicate an object is composed from one or more other objects. Association relationships indicate an object is related to one or more other objects in some way. OOD client-supplier relationhips define method calling protocols between objects. In this particular OOA/D notation, all relationships are binary i.e. link one object to one other object.

The EER notation defines named entities which have a collection of named attributes. Entities are linked by named relationships, each entity fulfilling a particular role in the relationship. The role links between an entity and a relationship may be named, may hold a cardinality (1:1, 1:N or M:N), and may have an optional or manditory flag. Subtyping relationships between entities have inclusive and exclusive flags.

The integrated OOEER conceptual data model integrates the notation of entity/attribute and object/attribute into a single entity/object notion. EER relationships and OOA/D relationships are integrated by being described as sub-types of a relationship notion, liked to entity/objects by a role. Figure 4 shows a conceptual data model integrating the OOA/D and EER notations.

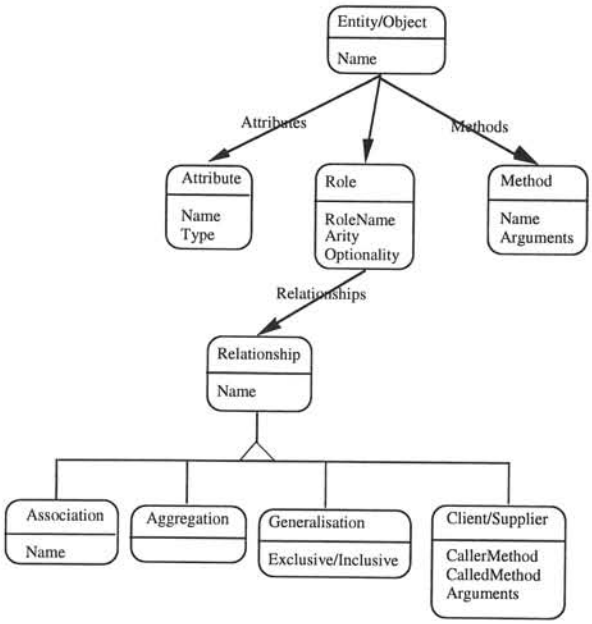

figure 4. An integrated data model for OOA/D and EER modelling.

5. Mappings Between Notation Data Models

There are some obvious mappings between OOA/D information and EER information, which we term *direct* mappings. These are illustrated in figure 5 by the relationship between corresponding EER entities and OOA/D objects. An entity corresponds directly to an object and an entity name to an object name, e.g. Customer entity to Customer object. Entity attributes also correspond directly to object attributes, e.g. Customer entity name to Customer object name. In an integrated OOA/D and EER environment, entities are described in the OOA/D notation by an object with the same name as the EER entity name. Similarly, object attribute names directly equate to entity attribute names. To maintain consistency between the notations, when an entity is added or deleted, an equivalently named object in the OOA/D notation is added or deleted, e.g. deleting the Customer entity means the Customer object is deleted. A similar direct translation applies between entity and object attributes.

Many CASE tools support these direct mappings. Many other mappings also exist between these notations, but these we term *indirect* mappings, i.e. an EER change can not be directly translated into an equivalent OOA/D change, and vice versa. For example, the OOA/D notation specifies the type of an attribute, such as string for Customer object name, but the EER notation does not. Changes to an attribute's type are thus either ignored by the EER notation or a description of the OOA change is presented to OOEER users. A similar approach can be used for object methods, such as the Customer set_credit method, which have no equivalent EER concept.

Creation and deletion of association relationships in the OOA notation, such as account-of between Customer and Account, can be directly translated into EER relationship addition and deletion. The reverse is not always true: if an EER relationship is created, this may be implemented by an OOA association or aggregation relationship. Such an inferred OOA relationship needs a change description documenting the inferal, so a user can appropriately make it an aggregation or association relationship. OOEER also provides a pop-up menu to allow designers to select the OOA relationhip they require.

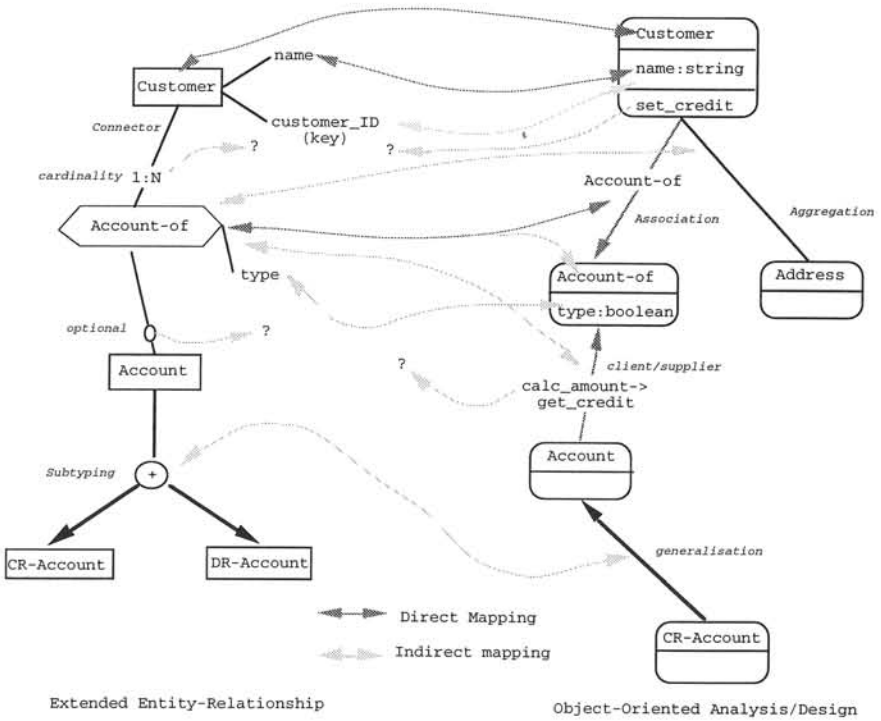

figure 5. Mappings between OOA/D concepts and EER concepts.

EER relationships of greater than binary cardinality (i.e. relating three or more entities) must be implemented by objects in the OOA/D notation. If an EER entity has attributes, then this relationship must also be represented as an OOA/D object, no matter what its cardinality. M:N relationships must be implemented in the OOA/D notation as an intermediate object with two 1:N collections. For example, if account-of between the Customer and Account entities has attributes or has arity M:N, then it must be implemented as an account-of object between Customer and Account objects.

EER subtyping relationships loosely correspond to OOA generalisation relationships and OOD inheritance relationships. The EER notation also has a notion of mutually inclusive subtypes and mutually exclusive subtypes, not supported in the OOA/D notation. A change to this mutually inclusive/exclusive state of an EER subtype relationship can't be directly reflected in the OOA/D notations, and a change description needs to be presented to users to document this indirect mapping. For example, if the subtyping relationship between Account and CR/DR-Account entities is changed to mutually exclusive, this can't be directly represented between the OOA/D Account and CR-Account objects.

The OOA/D notation has no concept of primary of foreign keys. However, changing an attribute in the OOA/D notation which corresponds to part of a primary or foreign key, such as the Customer customer_ID attribute, must be documented in the EER model, as it impacts on the semantic correctness of inter-entity relationships. If a relationship which is implemented by foreign key attribute(s) is deleted in either the EER or OOA models, an indication should be given to users that the attributes should be removed or ammended.

Further indirect mappings exist when translating changes between the EER and OOD notations. For example, the OOD client/supplier relationship, such as a method call by Account's calc_amount to Customer's get_credit method, may implement an EER relationship or may document a method call between two objects (not supported in the EER notation). Adding, removing or changing a client/supplier relationship can thus only be indicated by a change description in the EER model, with a user deciding on whether an EER relationship change is needed. EER 1:1 and 1:N relationships may be implemented as OOD object reference-typed attributes or as parameterised collection classes. Other mappings between the EER notation and OOD notation, and between the OOA and OOD notations, are not discussed further here but are supported by OOEER.

6. Tool Implementation and Integration

6.1. MViews

We have developed a framework of object-oriented classes, called MViews, which provides abstractions for implementing integrated ISEEs [2]. New environments are constructed by specialising framework classes to describe the data dictionary and program representation for ISEEs. Software system data is described by a graph-based structure, with graph *components* (nodes) specifying e.g. classes, entities, attributes and methods, and *relationships* (edges) linking these components to form the system structure. Multiple views of this data dictionary are defined using the same graph-based structure. These views are rendered and manipulated in concrete textual and graphical forms. External tools not built using MViews can be interfaced to the integrated environment by using extenal views.

Figure 6 shows an example of an integrated ISEE for object-oriented software development, SPE, developed using MViews [4]. The data dictionary describes classes, attributes and methods (collectively called "features"), inter-class relationships, and class and method implementation code. The multiple views provided by SPE include graphical OOA/D views and textual implementation and documentation views.

MViews supports very flexible inter-component consistency managenent by generating, propagating and responding to *change descriptions* whenever a component is modified. A change description documents the exact change in the state of a component and is propagated to all relationships the component participates in. These relationships can respond to this change description by applying operations to themselves or other components, forwarding the change description to related components, or ignoring the state change in the updated component. This technique supports a wide variety of consistency management facilities used by ISEE environments, including very flexible multiple view consistency, inter-component constraints, efficient incremental attribute recalculation, a generic undo/redo facility, and version control and collaborative facilities.

MViews is implemented in Snart, an object-oriented extension to Prolog. Environment implementers specialise Snart classes to define new environment data dictionaries, multiple views, and view renderings and editors. Snart is a persistent language, with objects dynamically saved and loaded to a persistent object store, making data dictionary and view persistency management transparent for ISEE implementers.

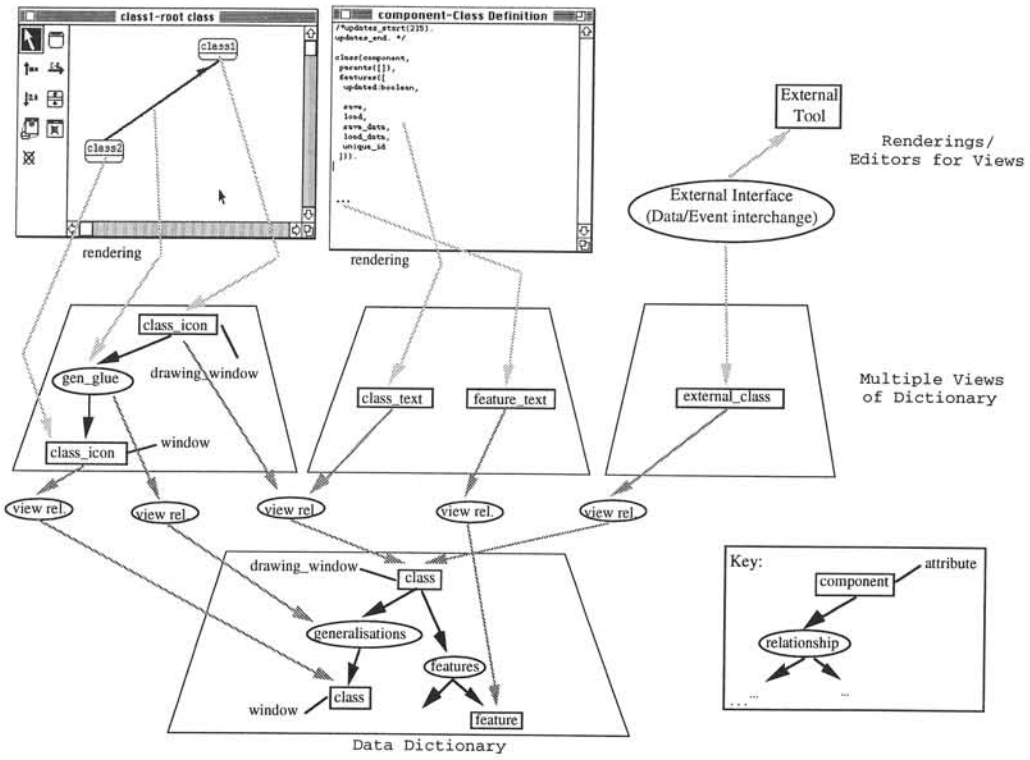

figure 6. Example of implementing an integrated ISEE using MViews.

6.2. Individual OOA/D and EER Tools

SPE (Snart Programming Environment) was implemented as a stand-alone environment for developing Snart programs [4]. It supports analysis, design, implementation, debugging and documentation of object-oriented programs using a variety of graphical and textual views. All of these views are kept consistent via the shared data dictionary, so information shown in one view is always consistent with other representations of the information in other views. This includes keeping analysis and design views bi-directionally consistent with implementation views, not supported by most other ISEEs and CASE tools. SPE has been used to model quite large object-oriented software applications, including complex architectural building model frameworks and the MViews and SPE frameworks themselves.

MViewsER was implemented as a stand-alone environment for EER modelling, which also supports textual relational schema views [3]. The graphical EER design diagram views are kept bi-directionally consistent with the textual relational schema views using the flexible MViews consistency mechanisms. MViewsER supports a basic EER notation and has been used to model a variety of Information System problems, with the relational schema exported to relational database environments for implementation.

6.3. The Integrated OOEER Environment

The conceptual data models used in the construction of SPE and MViewsER equate to those defined in Section 4. We have integrated SPE and MViewsER into one integrated environment, OOEER, which supports both the SPE OOA/D notation and the MViewsER EER notation. OOEER also provides textual views for implementing an object-oriented program and relational schema based on these design notations. All of these views are kept consistent by the environment. As noted in section 3, some of these changes are automatically carried out by OOEER, while for others change descriptions are displayed to users for manual implementation.

SPE and MViewsER were integrated to produce OOEER by defining a data dictionary based on the integrated conceptual data model defined in section 4. The mappings defined in section 5 were used to link the different components and relationships in each notation's data dictionary, as shown by the light lines in figure 7. The mappings were also used to define translations for change descriptions generated by each environment's data dictionary into updates on the integrated data dictionary and then updates on the other notation's data dictionary, shown by the thick lines in figure 7.

The change descriptions generated by SPE must be translated into updates on the integrated data dictionary and then into appropriate updates on MViewsER's data dictionary. The integrated data dictionary components are related to those in the SPE data dictionary, and thus the inter-dictionary relationships are sent change descriptions when SPE data changes. These relationships respond to these change descriptions to update the integrated data dictionary state, thus implementing direct mappings. For indirect mappings, default changes are made to the integrated data dictionary (where possible) and the change descriptions stored, so they can be displayed in views for users to check the correct update was inferred.

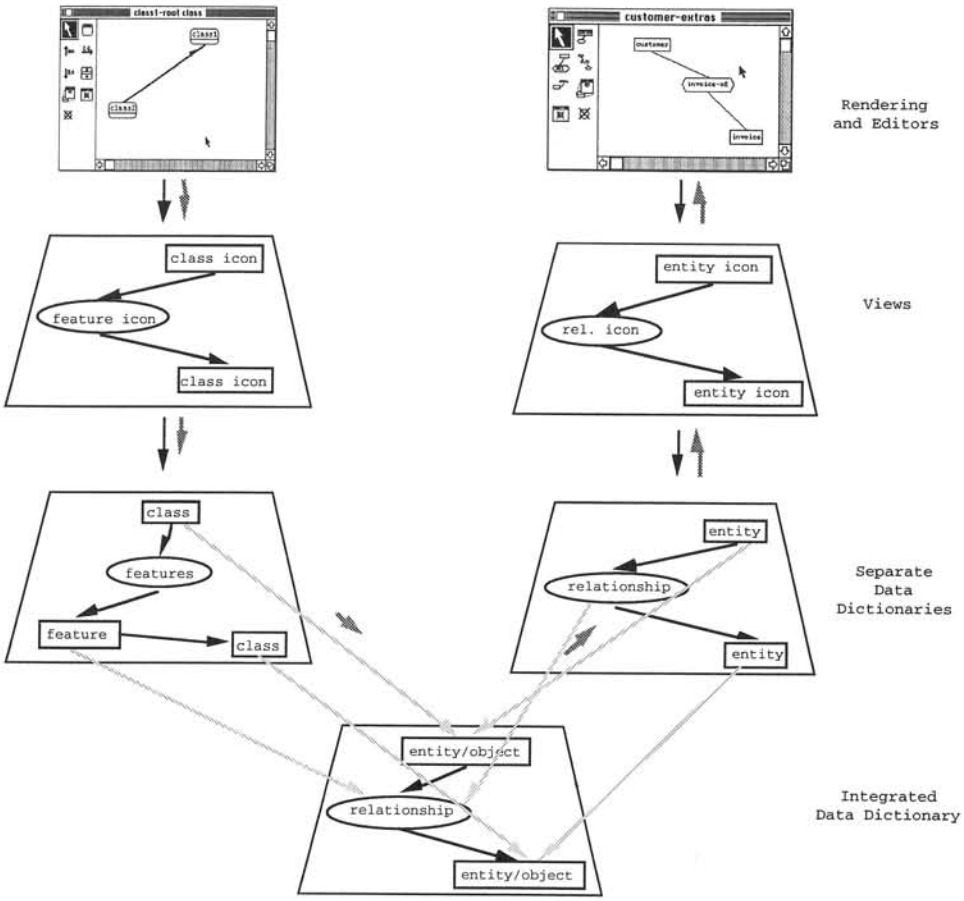

figure 7. Integrating SPE (OOA/D) and MViewsER (EER) ISEEs using the integrated data model.

When the integrated data dictionary components change, they broadcast change messages to the inter-dictionary relationships linking MViewsER's data dictionary components to the integrated dictionary. These relationships translate the change descriptions on the integrated dictionary components into updates on the MViewsER components. Indirect mapping changes may be defaulted where possible, but the change descriptions are also displayed in views. Both SPE and MViewsER keep their multiple views consistent; the inter-dictionary mappings keep their underlying dictionary data consistent.

Neither SPE nor MViewsER were modified in any way to support this integration process to produce OOEER. Change descriptions from another notation are displayed as special MViews "user updates" in the other notation's tools. This meant no special display mechanisms had to be added to SPE or MViewsER so they could display these new change descriptions. Some interesting semantic conflicts did occur during integration. For example, adding an entity attribute in MViewsER which has the same name as a class method in SPE will produce a semantic error when the change is propagated to SPE (as all class attributes and methods must have unique names). OOEER detects this and similar semantic errors during translation to the integrated data dictionary and produces special MViews "error" change descriptions to document the inter-notation semantic conflict.

6.4. Experience

We have used OOEER to model several small-to-medium Information System designs. It provides support for not only direct mappings between the OOA/D and EER notations, but more importantly the indirect mappings, allowing both notations to be more effectively used on the same problem domain. When

working with a particular notation's view, users are informed of any related changes in both other views for the notation and views for the other notation. Direct mapping changes are automatically made, and change descriptions not shown, while indirect mapping changes can be attempted by the environment and are always documented.

MViews provides lazy processing capabilities which we have used to minimise the response time delay for users. Much of the translation and consistency management is done on-demand when a view is selected for editing by caching change descriptions in the integrated data dictionary. When a view from a different tool is to be edited, MViews informs the integrated data dictionary which then actions any cached change descriptions. This results in a minimum affect of tool response time when making discrete view edits, and delays inter-notation translation until actually required. The resulting performance of the integrated environment is not significantly different to the individual MViewsER and SPE environments.

A major advantage of the integrated data dictionary over a direct mapping between the OOA/D and EER notations is for environment extensibility. For example, if a NIAM notation modelling tool were to be added to OOEER, the concepts of the NIAM notation which directly and indirectly relate to those in the other models are related via the integrated data model. This reduces the number of mappings which have to be specified, as many translations, particularly the direct ones, are already implemented. This also makes the individual tools easier to extend, as the tool's data dictionary can be extended with little affect on the integrated dictionary and, more importantly, on the inter-notation mappings. The integrated data dictionary also provides a useful source for hypertext links between views for different notations. Rather than just allowing a user to navigate around the views for a particular notation, the integrated data model and its inter-dictionary relationships are used by OOEER to allow users to access the views in other models which contain corresponding concepts.

7. Conclusions

We have developed a new method of integrating different design notations within Information Systems Engineering Environments. The conceptual data models of different design notations are defined and then an integrated data model derived, together with mappings of concepts and data changes between each notation's data model. Environments supporting each individual notation are then implemented based on this design by reusing the MViews framework. The separate environments are integrated into one environment by implementing an integrated data dictionary based on the integrated conceptual data model. The concept and data change mappings are then used to link related data from each notation's dictionary, and to keep these dynamically consistent as they change. We have developed OOEER, an ISEE which supports integrated OOA/D and EER notations using this approach. OOEER propagates direct changes between the OOA/D and EER notation views, such as entity, object and attribute creation, renaming, and deletion. It also propagates indirect changes, such as adding and renaming EER relationships and adding and changing OOA/D inheritance, aggregation and association relationships, which are not supported by most CASE tools.

We are currently extending OOEER to use the data model mappings to support decision tracability, not only between analysis and design notations but between different analysis and design notations. This will allow users to trace the reasons analysis and design decisions were made through each notation's views and down to implementation views. We are also extending OOEER to support version control for analysis and design views and Computer-Supported Co-operative Work facilities [5]. This will allow multiple users to collaborate on system analysis and design using not only multiple views but also multiple notations. The inter-notation mapping technique is also being used to support intra-notation mapping in SPE i.e. mapping between analysis and design concepts and keeping these consistent under change. We are also extending OOEER to support user-defined links between differently-named classes and entities, and their relationships. This will allow users to specify different structures in the EER and OOA/D notations, which can then be manually linked, and thus the environment can help keep them consistent.

References

[1] Chen, P.P., " The Entity-Relationship Model - Toward a Unified View of Data," *ACM Transactions on Database Systems*, vol. 1, no. 1, 9-36, 1976.

[2] Grundy, J.C. and Hosking, J.G., "A framework for building visusal programming environments," in *Proceedings of the1993 IEEE Symposium on Visual Languages,* IEEE Computer Society Press, 1993, pp. 220-224.

[3] Grundy, J.C., "Multiple textual and graphical views for Interactive Software Development Environments," Ph.D. thesis, University of Auckland, Department of Computer Science, June 1993.

[4] Grundy, J.C., Hosking, J.G., Fenwick, S., and Mugridge, W.B., *Visual Object-Oriented Programming.* Prentice-Hall, 1994, chap. 11.

[5] Grundy, J.C., Mugridge, W.B., Hosking, J.G., and Amor, R., "Support for Collaborative, Integrated Software Development," accepted to *the 7th Conference on Software Engineering Environments,* 1995.

[6] Henderson-Sellers, B. and J.M., E., "The Object-Oriented Systems Life Cycle," *Communications of the ACM*, vol. 33, no. 9, 142-159, 1990.

[7] Heym, M. and Ôsterle, H., " A Semantic Data Model for Methodology Engineering," in *Proceedings of the Fifth International Workshop on Computer-Aided Software Engineering,* IEEE Computer Society Press, Washington, D.C., 1992, pp. 142-155.

[8] Nuseibeh, B. and Finkelstein, A., "ViewPoints: A Vehicle for Method and Tool Integration," in *Proceedings of the Fifth International Workshop on Computer-Aided Software Engineering,* IEEE Computer Society Press, Washington, D.C., 1992, pp. 50-61.

[9] Ratcliffe, M., Wang, C., Gautier, R.J., and B.R., W., " Dora - a structure oriented environment generator," *IEE Software Engineering Journal*, vol. 7, no. 3, 184-190, 1992.

[10] Reiss, S.P., " Connecting Tools Using Message Passing in the Field Environment," *IEEE Software*, vol. 7, no. 7, 57-66, July 1990.

[11] Song, X. and Osterweil, L.J., "A Process-Modeling Based Approach to Comparing and Integrating Software Design Methodologies," in *Proceedings of the Fifth International Workshop on Computer-Aided Software Engineering,* IEEE Computer Society Press, Washingon, D.C., 1992, pp. 225-229.

[12] *TurboCASE Reference Manual*, StructSoft Inc, 5416 156th Ave. S.E. Bellevue, WA, 1992.

[13] Venable, J.R., "CoCoA: A Conceptual Data Modelling Approach for Complex Problem Domains," Ph.D. thesis, Thomas J. Watson School of Engineering and Applied Science, State University of New York at Binghampton, 1993.

[14] Wasserman, A.I. and Pircher, P.A., " A Graphical, Extensible, Integrated Environment for Software Development," *SIGPLAN Notices*, vol. 22, no. 1, 131-142, January 1987.