Providing Integrated Support for Multiple Development Notations

John C. Grundy and John R. Venable

Department of Computer Science, University of Waikato
Private Bag 3105, Hamilton, New Zealand
email: jgrundy@cs.waikato.ac.nz or jvenable@cs.waikato.ac.nz

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. Our 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, and then merged into an integrated conceptual data model. Second, mappings are derived from the integrated conceptual data model, which translates 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 repository 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 to one system description are propogated to all other system descriptions, using the same or different notations.

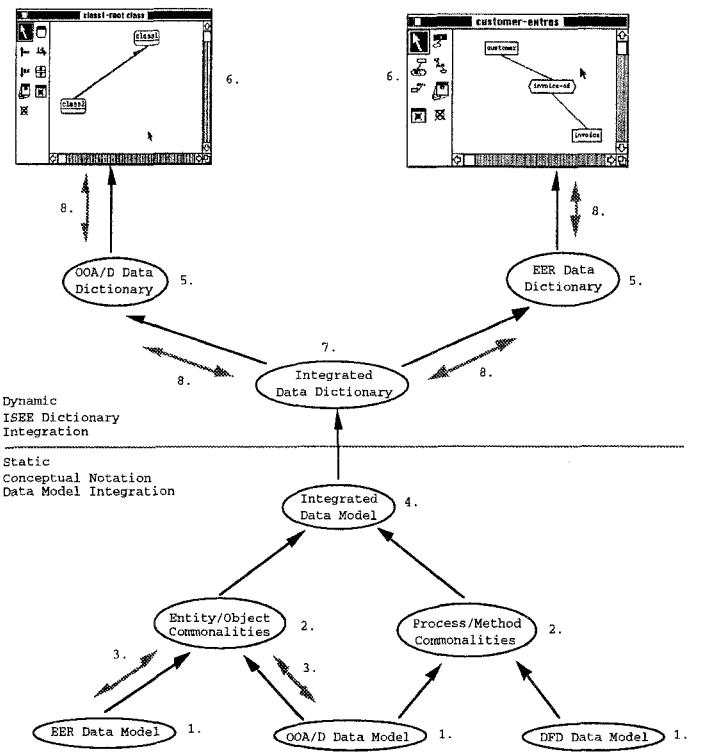

Fig. 1. Integrating design notations via a hierarchical data model.

We present a new approach which supports both static and dynamic integration of multiple notations, illustrated in Fig. 1. The first 4 steps achieve 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) Dynamic mappings between different notation components are identified i.e. how changes to data in one model are translated into changes to the other. 4) Several levels of partial data models are built up and collected to describe an integrated data model for all of the design notations.

The second 4 steps achieve dynamic integration of design and implementation tools for the different notations, with the conceptual data models and mappings used to implement an ISEE encorporating all of the notations. 5) Individual tools for each notation are developed with the tool repository based on the notation's conceptual data model. 6) Editors for this repository are developed, including multiple textual and

graphical notation views. 7) The integrated data model (and possibly the intermediate partial data models) are used to define an integrated repository. 8) Data changes in one notation's repository are propagated to other notation repositories via the integrated repository, with change translation as specified by the mappings defined in step 3.

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 individual OOA/D and EER notations, and an integrated conceptual data model, is briefly described with mappings between the notations. Two previously developed OOA/D and EER environments are introduced and the implementation of OOEER by integrating 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 designers to analyse, design, and implement Information Systems from within one environment, providing a consistent user interace and consistent repository (data dictionary). They help to minimise inconsistencies that can arise when using several separate tools for information systems development [17, 13].

Some work has been done on the static integration of notations. Wieringa [18] has compared JSD, ER modelling and DFD modelling, and examined the possibilities of integrating different aspects these methods. Data modelling has been used to compare different notations [11] and support methodology engineering [10]. Process-modelling has also been applied to compare and integrate notations [14]. There has been little work at developing detailed integration of individual notations, and little has been done to translate conceptually integrated notations into tool-based implementations.

Limited dynamic notation integration is supported by many CASE tools, such as Softare thru Pictures™ [17] and TurboCASE™ [15]. These ICASE environments allow developers to analyse and design software using a variety of different notations, with limited inter-notation consistency. For example, entities from an EER model can be used as objects in an OOA model, with name and attribute changes kept consistent. Such tools do not generally support 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, and consistency between design and implementation code is often not maintained.

FIELD environments [13] utilise selective broadcasting, involving propagating messages between separate Unix tools, to keep muliple tool views conistent. FIELD can not effectively integrate different design notation tools, as it provides no integrated repository. As more tools are added the translation process becomes complex and difficult to implement. Dora [12] provides abstractions for keeping multiple textual and graphical views consistent under change. These views share the same repository and hence can more easily be kept consistent. Dora does not provide any mechanism for propagating changes between views which can not be directly applied by the environment. Thus some changes made to a design which can not be automatically implemented in another notation by the environment can not be supported.

Two key methods are thus required for dynamic integration: (1) utilising an integrated repository and (2) providing additional support for ensuring consistency

between aspects which cannot be automatically kept consistent by the environment. 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 and change mappings between the notation's conceptual 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.

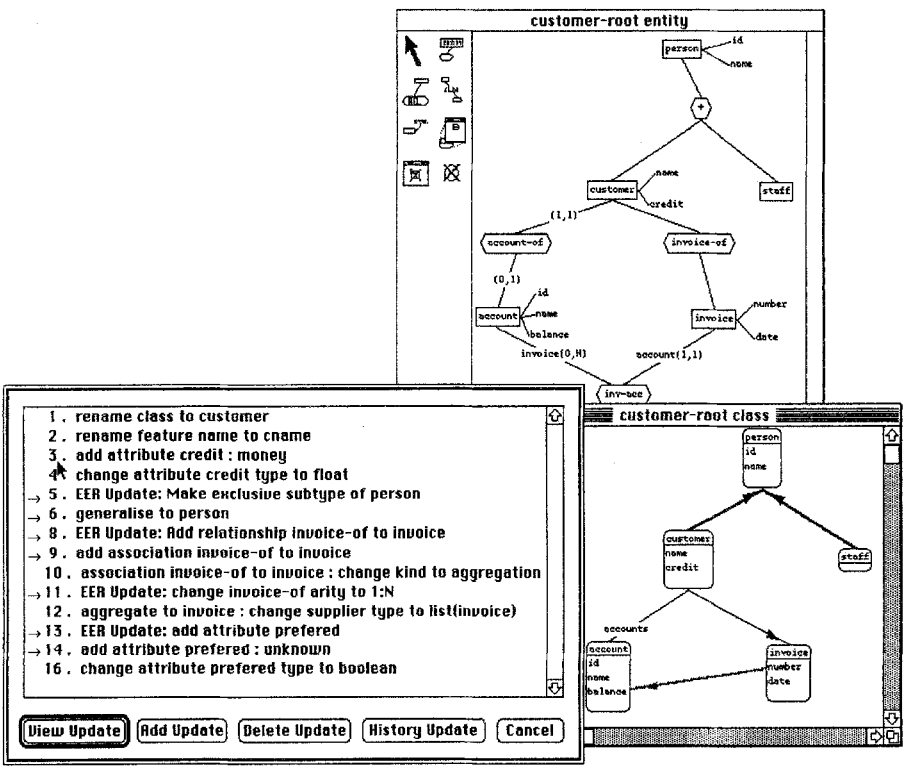

Fig. 2. The integrated OOEER environment.

A screen dump from OOEER is shown in Fig. 2. This shows an OOA view ('customer-root class') and an EER view ('customer-root entity') for an invoicing Information System. The dialog shows the modification history for the customer

class. Items highlighted with a '→' were actually made to the EER customer entity, and were automatically translated to the OOA customer class by OOEER.

The OOA/D notation is based on [9]. 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 arguments. The EER notation is based on Chen's ER model [1]. Square icons represent entities with optional named attributes connected to the icon. Diamond-shaped icons represent relationships with optional role names and arities on the connecting lines. Subtyping is exclusive ('-') or inclusive ('+').

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 (OOA/D, EER and implementation views). Where the environment can automatically make an appropriate change to keep these 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, however, 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. In Fig. 2, change #8 was made to the EER view, while OOEER autoamtically made change #9 to the OOA view. The designer can view the description(s) of the EER view change(s) made which affect the OOA view, and which could not be fully implemented by the environment. The designer then manually changes the new OOA relationship to an aggregation relationship (change #10).

Fig. 2 shows some other EER changes on the customer entity and its relationships, which have been (semi-)automatically translated into OOA view updates. Some changes made in the EER view require the designer to further modify the OOA view, to ensure it is correct. For example, making a customer entity an exclusive subtype of the person entity (change #5) is directly translated into generalising the customer class to the person class (change #6). Changing the arity of a relationship (change #11), can not be automatically implemented by OOEER. Instead, the designer is informed of the EER change by the presence of this change description. The designer then must manually change the type of the aggregation relationship between the customer and invoice classes to list(invoice) (change #12), to implement this OOA/D relationship. Similarly, if an untyped attribute is added to the customer entity (change #13), OOEER translates this into an attribute addition to the customer class (change #14). The designer then must manually define an appropriate attribute type (change #15).

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, as shown in Fig. 3. These inform programmers of changes that need to be made to keep the implementation view consistent. 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. In total OOEER

supports fully integrated OOA, OOD, EER and debugging graphical views, and class definition, class contract, method implementation, relational schema and general documentation textual views.

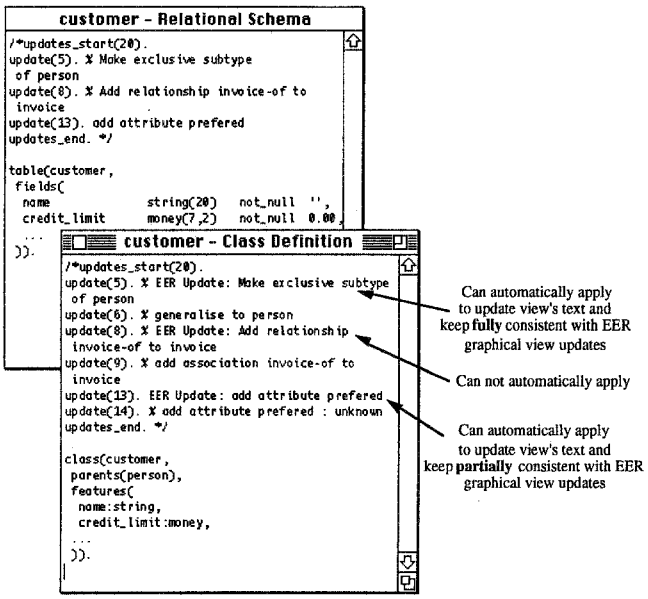

Fig. 3. OOEER textual view consistency.

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 [16]. Fig. 4 shows the conceptual data models for OOEER's EER and OOA/D notations.

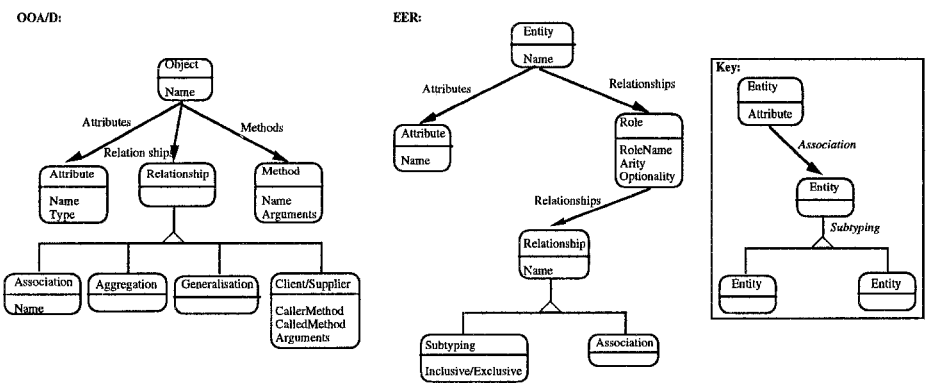

Fig. 4. Separate conceptual data models for OOA/D and EER notations.

The OOA/D notation has named objects, with a collection of named/typed attributes and methods with arguments. Generalisation relationships indicate an object is generalised to another object, aggregation relationships indicate an object is composed of other object(s), and association relationships indicate an object is related to other object(s) in some way. Client-supplier relationhips define method calling protocols.

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, as shown in Fig. 5.

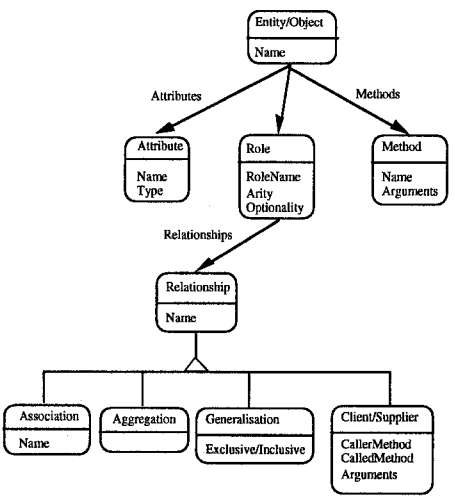

Fig. 5. An integrated data model for OOA/D and EER modelling.

5 Mappings Between Notation Data Models

There are some simple mappings between OOA/D information and EER information, which we term *direct* mappings. These are illustrated in Fig. 6. An entity corresponds directly to an object and an entity name to an object name. Entity attributes also correspond directly to object attributes. in OOEER, EER entities are thus described by an OOA/D object with the same name as the entity name. Similarly, object attribute names directly equate to entity attributes. To maintain consistency, when an entity/attribute is added or deleted, a same-named object/attribute is added or deleted.

Many CASE tools support these direct mappings. Many other mappings also exist between these notations, which we term *indirect* mappings, i.e. an EER change can be translated to the integrated repository, but the resulting change on the integrated model cannot be directly translated to the OOA/D model, and similar for some OOA/D to EER changes. For example, the OOA/D notation specifies the type of an attribute 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/D change is presented to users. A similar approach is used for object methods, which have no EER concept.

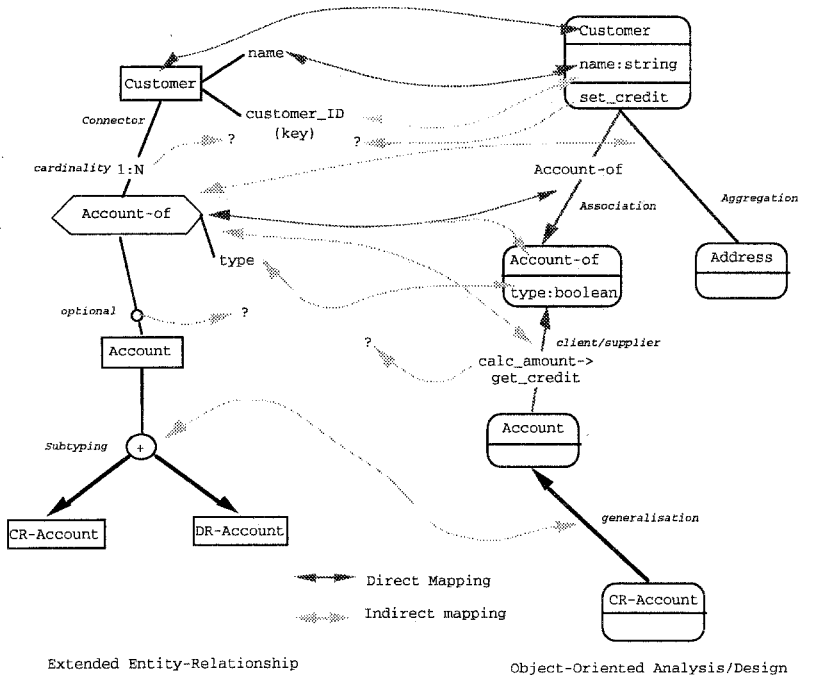

Fig. 6. Mappings between OOA/D concepts and EER concepts.

Creation and deletion of OOA/D association relationships 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 designer can appropriately make it an aggregation or association relationship. EER relationships of greater than binary cardinality, or with attributes, must be implemented by OOA/D objects, as must an EER M:N relationship, via an objet and two 1:N connections.

EER subtyping relationships loosely correspond to OOA/D generalisation/inheritance relationships. Mutually inclusive and exclusive EER subtypes are not, however, supported in the OOA/D notation. A change to this inclusive/exclusive state of an EER subtype thus can't be directly reflected in OOA/D relationships, and a change description is needed to document this indirect mapping.

The OOA/D notation has no concept of EER primary or foreign keys. Changing an OOA/D attribute which corresponds to part of a EER primary or foreign key must be documented, as it impacts on the EER model semantic correctness. If a relationship implemented by foreign key(s) is deleted, an indication should be given to the designer that EER attributes should be removed or ammended. A notation's formal semantics must thus be considered when determining automatic and semi-automatic translations.

Further indirect mappings exist when translating changes between the EER and OOD notations. The OOD client/supplier relationship may implement an EER relationship or may document a method call between two objects. Adding, removing

or changing a client/supplier relationship can thus only be indicated by a change description in the EER model. The designer decides on whether an EER relationship change is needed. EER 1:1 and 1:N relationships may be implemented as OOD reference-typed attributes or as generic collection classes. Various other mappings between the EER notation and OOD notation, are not discussed further here but are supported by OOEER.

6 Tool Implementation and Integration

6.1 MViews

We have developed MViews, which provides abstractions for implementing integrated ISEEs [3, 5]. New environments are constructed by specialising object-oriented framework classes to describe the repository 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 repository are defined using the same graph-based structure. These views are rendered and manipulated in concrete textual and graphical forms.

Fig. 7 illustrates the structure of SPE, an integrated ISEE for object-oriented software development, developed using MViews [6]. The repository describes classes, attributes and methods (features), inter-class relationships, and implementation code. SPE multiple views include graphical OOA/D and textual implementation and documentation views.

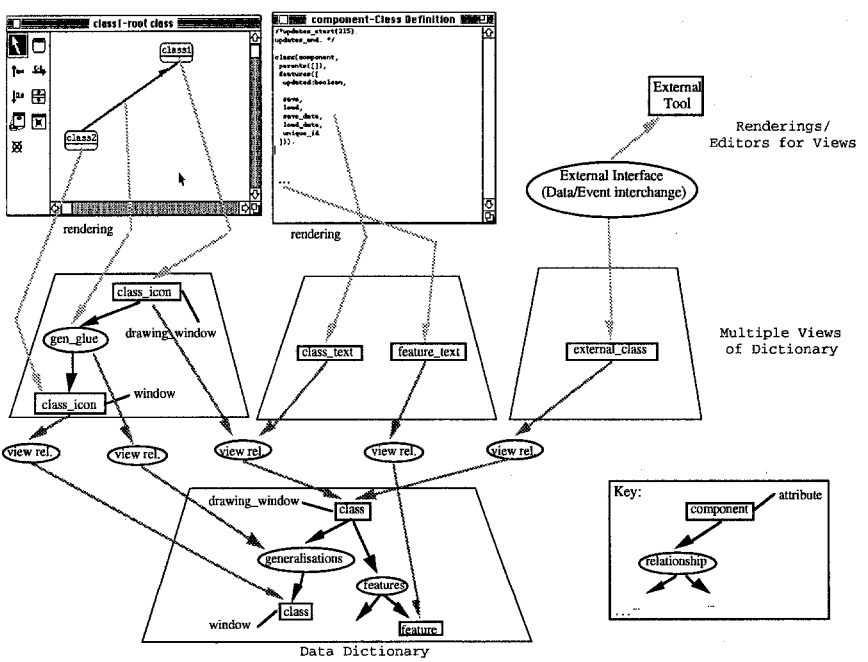

Fig. 7. Example of implementing an integrated ISEE using MViews.

MViews components support 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 respond to change descriptions by applying operations to themselves or other components, forwarding the description to related components, or ignoring the component update. Most ISEE consistency management facilities are supported by this technique, including flexible multiple view consistency, constraints, attribute recalculation, undo/redo, and version control and cooperative work [7].

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 repository and view persistency management transparent for ISEE implementers. External tools not built using MViews can be interfaced to the integrated environment by using extenal views.

SPE was implemented as an ISEE for developing Snart programs [6]. It supports analysis, design, implementation, debugging and documentation of object-oriented programs using graphical and textual views. Views are kept consistent via the shared repository, so information in one view is always consistent with other representations in other views. This includes keeping analysis and design views bi-directionally consistent with implementation views, not supported by most ISEEs. SPE has been used to model large object-oriented software applications, including architectural building model frameworks and the MViews and SPE frameworks.

MViewsER was implemented as an ISEE for EER modelling, and also supports textual relational schema views [4]. Graphical EER design views are kept bi-directionally consistent with textual relational schema views. MViewsER has been used to model a variety of Information System problems, with the relational schema exported to relational database environments for Information System implementation.

6.2 The Integrated OOEER Environment

The conceptual data models used in the construction of SPE and MViewsER more or less equate to those defined in Section 4. We have integrated SPE and MViewsER into one integrated environment, OOEER, which supports integrated OOA/D and EER design and implementation. All of these views are kept consistent by the environment. As noted in section 3, some of these changes are (partially) automatically carried out by OOEER, while for others change descriptions are displayed to designers for manual implementation.

SPE and MViewsER were integrated by defining a repository based on an integrated conceptual data model (section 4). Mappings (section 5) were used to link the components and relationships in each notation's repository. The mappings also define translations for change descriptions generated by each environment's repository into updates on the integrated repository, then updates on the other notation's repository.

When an SPE view is edited (1), the modification is translated into SPE repository updates (2), generating change descriptions. The inter-repository relationships are sent change descriptions, and respond to these by updating the integrated repository (3). When the integrated repository components change, the inter-repository relationships to MViewsER's repository components translate the integrated repository components

change descriptions into updates on MViewsER repository components (4). Indirect mapping changes are defaulted where possible and change descriptions displayed in views. Both SPE and MViewsER keep their multiple views consistent (5 and 6).

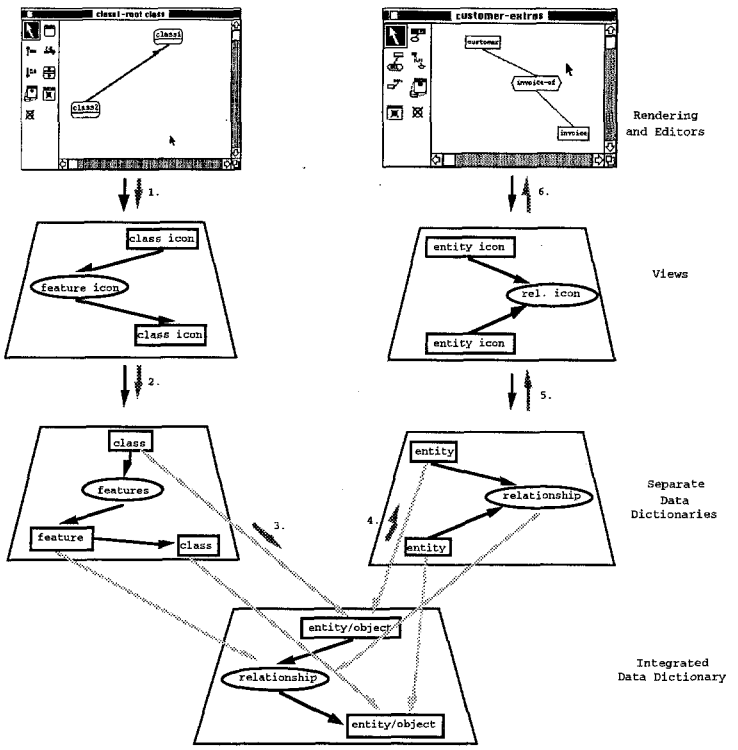

Fig. 8. Integrating SPE and MViewsER using the integrated data model.

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, requiring no special display mechanisms in the other tool. Any semantic errors detected during translation from the integrated repository are documented with "error" change descriptions.

6.3 Inter-repository Relationships

Inter-repository relationships are implemented as specialisations of MViews' generic many-to-many relationships. This allows one or more components from one repository to be connected to one or more components in another repository. When a change description from one component participating in the relationship is received, the relationship component determines the appropriate change to make to other participating components. This might be a simple update (automatic translation), a partial update (semi-automatic translation) or simply storing the change description against the affected component(s) (no automatic translation by OOEER possible). Using MViews' change description composition facilities [7], the inter-repository relationships can even wait for several change descriptions to be received and then translate them into changes on other related components.

We used MViews' lazy processing capabilities to minimise response time delay for users. Much of the update translation and view consistency management is done on-demand when a view is selected for editing. Change descriptions are cached in the integrated repository, and when a view from a different notation is to be edited, the integrated repository actions any cached change descriptions. This results in a minimum affect on tool response time when making discrete view edits.

6.4 Experience

We have used OOEER to model several small-to-medium Information System designs. As both direct and indirect mappings between OOA/D and EER notations are supported, both notations can be more effectively used on the same problem domain. When working with a view, designers are informed of any related changes in both other views for the notation and views for the other notation.

A major advantage of an integrated repository over a direct mapping between notations is for environment extensibility. For example, if a NIAM notation 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. Individual tools are also easier to extend, as the tool's repository can be extended with little affect on the integrated repository or on the inter-notation mappings. The integrated repository also provides a useful source for hypertext links between views for different notations.

Currently inter-repository relationships are automatically created by OOEER. We are currently extending OOEER to support user-defined relationships between different notation components, to allow a designer to relate one (or more) items in an OOA/D model to one (or more) items in an EER model. Limited consistency management across these relationships will be supported, mainly informing designers when a component on one side of the relationship has been altered.

In this work we have considered mappings between graphical icon-and-glue and textual OOA/D and EER notation components. Spatial constructs, such as Coad and Yourdon subjects [2] can be implemented, if desired. We are designing new tools to support NIAM diagrams, state transition diagrams and data flow diagrams, which will be integrated into OOEER. These different notations can be kept partially consistent with OOA/D and EER views using our technique. Some inter-notation consistency issues are more difficult to implement than others for these notations, and for some limited consistency can only be provided via user-defined relationships.

7 Conclusions

We have developed a new method for integrating different design notations within ISEEs. 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 data model. Tools supporting each notation are implemented based on this design by reusing the MViews framework. These separate tools are integrated by implementing an integrated repository based on the integrated conceptual data model. The concept and data change mappings are used to link related data from each notation's repository, 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, not supported by most CASE tools.

We are extending OOEER to use data model mappings to support decision tracability between both analysis and design notations and different design notations. This will allow designers to trace analysis and design decisions through each notation's views and to implementation views. We are also extending OOEER to support version control for analysis and design views and Computer-Supported Co-operative Work facilities, as done for SPE [8]. Multiple designers will be able to collaborate on analysis and design using both multiple views and multiple notations. An issue is maintaining consistency between different notation views shared by designers. Our inter-notation mapping technique is being used to support intra-notation mapping in SPE i.e. mapping between analysis and design concepts and keeping these consistent under change. User-defined links between differently-named classes and entities, and their relationships, will allow designers to specify different EER and OOA/D structures, which can be manually linked and partially keep consistent by OOEER.

References

1. P.P. Chen: The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems 1*, 1 (1976), 9-36.
2. P. Coad and E. Yourdon: *Object-Oriented Analysis,* Yourdon Press, Second Edition (1991).
3. J.C. Grundy, and J.G. Hosking: A framework for building visusal programming environments. In *Proceedings of the 1993 IEEE Symposium on Visual Languages,* IEEE Computer Society Press, 1993, pp. 220-224.
4. J.C. Grundy: *Multiple textual and graphical views for Interactive Software Development Environments*, Ph.D. dissertation, University of Auckland, Department of Computer Science, June 1993.
5. J.C. Grundy and J.G. Hosking: Constructing Integrated Software Development Environments with Dependency Graphs, Working Paper, Department of Computer Science, University of Waikato, 1994.
6. J.C. Grundy, J.G. Hosking, S. Fenwick, and W.B. Mugridge:*Visual Object-Oriented Programming,* Prentice-Hall (1994), Chapter 11.
7. J.C. Grundy, J.G. Hosking, and W.B. Mugridge, Supporting flexible consistency management via discrete change description propagation, Working Paper, Department of Computer Science, University of Waikato, 1995.
8. J.C. Grundy, W.B. Mugridge, J.G. Hosking, and R. Amor: Support for Collaborative, Integrated Software Development. accepted to the *7th Conference on Software Engineering Environments,* IEEE CS Press, April 1995.
9. B. Henderson-Sellersand J.M. Edwards: The Object-Oriented Systems Life Cycle. *Communications of the ACM 33*, 9 (1990), 142-159.
10. M. Heym and H. Österle: A Semantic Data Model for Methodology Engineering. In *Proceedings of the Fifth International Workshop on Computer-Aided Software Engineering,* IEEE CS Press, Washington, D.C., 1992, pp. 142-155.

11. B. Nuseibeh and A. Finkelstein: ViewPoints: A Vehicle for Method and Tool Integration. In *Proceedings of the Fifth International Workshop on Computer-Aided Software Engineering,* IEEE CS Press, Washington, D.C., 1992, pp. 50-61.
12. M. Ratcliffe, C. Wang, R.J. Gautier, and B.R. Whittle: Dora - a structure oriented environment generator. *IEE Software Engineering Journal 7*, 3 (1992), 184-190.
13. S.P. Reiss: Connecting Tools Using Message Passing in the Field Environment. *IEEE Software 7*, 7 (July 1990), 57-66.
14. X. Song, and L.J. Osterweil: 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.
15. *TurboCASE Reference Manual*, StructSoft Inc, 5416 156th Ave. S.E. Bellevue, WA, 1992.
16. J.R. Venable:*CoCoA: A Conceptual Data Modelling Approach for Complex Problem Domains*, Ph.D. dissertation, State University of New York at Binghampton, 1993.
17. A.I. Wasserman, and P.A. Pircher: A Graphical, Extensible, Integrated Environment for Software Development. *SIGPLAN Notices 22*, 1 (January 1987), 131-142.
18. R.J. Wieringa: Combining static and dynamic modelling methods: a comparison of four methods. *to appear in Computer Journal* (1995).