Revised version in Proceedings of the 6th Workshop on the Next Generation of CASE Tools, Finland, June 1995.

Developing CASE tools which support integrated development notations

John C. Grundy and John R. Venable

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

This paper discusses the recent work of the authors in developing Integrated CASE (ICASE) tools which support multiple development notations. We use a four-step methodology to develop these integrated tools. The conceptual data models of different notations are first developed, and then merged into an integrated conceptual data model. Dynamic mappings between different notation components are used to describe how related notation components are kept consistent under change. The conceptual data models and mappings are used to implement individual CASE tools for each notation. Separate tools are then integrated by linking their repositories via a shared repository based on the integrated conceptual data model. We discuss extensions to this work which will provide MetaCASE facilities for environment generation and evolution.

1. Introduction and Motivation

This paper briefly summarises ongoing and planned research work at the University of Waikato to develop methods and architectures for developing integrated CASE tool sets. Several aspects of CASE are of interest to us, including:

- Integration and support of multiple information systems development (ISD) and software engineering (SE) methods and notations [6]
- Methodology engineering [11]
- Support for cooperative work aspects of CASE tool use [7]
- Automatic support for propagation of changes between integrated CASE tools [2, 6]
- MetaCASE environments for specifying and generating integrated CASE tools

This paper concentrates on the first and last two of these areas of interest. It addresses our interest in providing integrated support for multiple methods and notations, an important aspect of that support, i.e. propagation of changes between tools, and our vision for a MetaCASE tool environment for generating integrated CASE tools.

2. Integrated CASE Tool Environment Development Method

We have developed a four-step method for supporting the development of integrated CASE tool environments, as illustrated in figure 1:

1. *Conceptual ISML Integration.* Static data modelling and integration of the concepts of the Information Systems Modelling Languages (ISMLs) to be supported by the CASE tool set.
2. *Inter-ISML Mappings.* Specification and inference of the dependency links and dynamic mappings within and between conceptual ISML components.
3. *Tool Implementation.* For each tool, selection of the concepts relevant to the ISML that the tool will support, and implementing the selected concepts as a tool repository and appropriate ISML notation editors.
4. *Tool Integration.* Specification of mechanisms for implementing the inter-tool updating and/or notification caused by changes made in one of the other tools. User and/or tool designer definition of additional inter-tool dependency links.

Thus far, this method is mostly manual, but makes use of a framework and an extensive library of classes for the implementation of integrated CASE tool environments which support multiple notations [2, 4]. However, we are designing an integrated MetaCASE environment for the specification and generation of integrated CASE tool environments, which will support the development approach described in this paper.

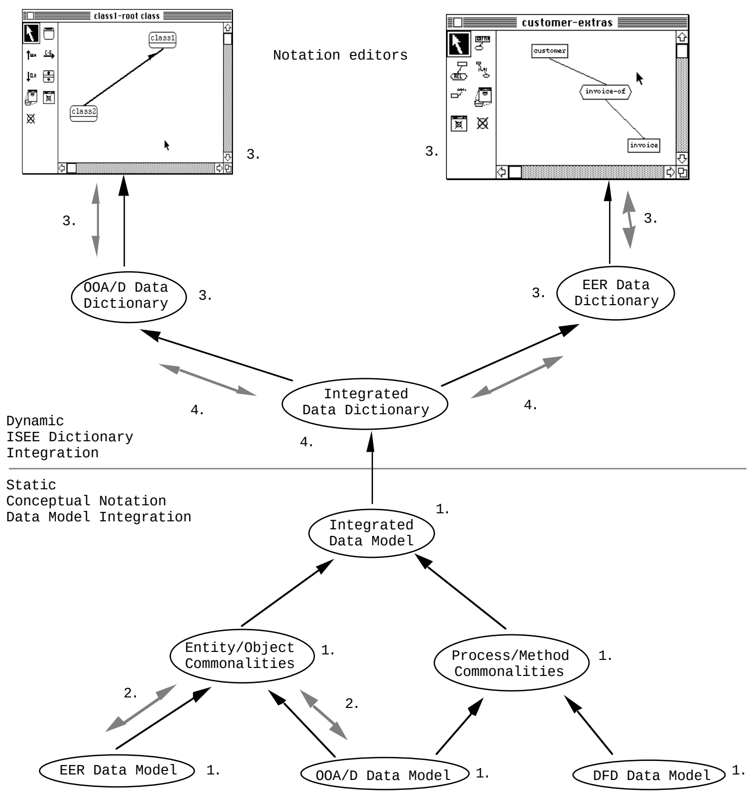


figure 1. Our approach to building integrated CASE tools.

3. Conceptual ISML Integration

In order to develop an integrated set of CASE tools, the ISMLs to be embodied in the tools must be integrated. The relationships, particularly the overlaps, between the ISMLs must be made clear. To do so, we perform static conceptual data modelling of the individual ISMLs and view integration between different ISMLs using the conceptual data modelling language, CoCoA (named for Complex Covering Aggregation) [11]. First, the individual ISMLs are modelled as precisely as possible. Second, the individual data models, which represent the views of the different ISMLs, are integrated. This second step typically involves resolution of synonyms (by selecting a single, unified name) and homonyms (by changing or specialising one of more of the names), as well as the introduction of various superclasses and intervening subclasses.

Figure 2 gives an example of integrated views, that of the integration of the views of the SADT Activity Diagram, DFD, and ISAC A-graph ISMLs. Using CoCoA, it is possible to integrate any number of different ISMLs' views, whether those ISMLs' views are static, dynamic, state- oriented, or whatever. For example, it is possible to data model the integration of JSD, ER, and DFD Modelling proposed by Wieringa [13].

The resulting data models serve as specifications for the multiple, hierarchically integrated CASE tool repositories used in our architecture for integrating the tools. Each individual ISML's CoCoA diagrams serve as a specification for the repository for the individual CASE tool that supports that ISML. The integrated CoCoA model serves as the specification for the integrated repository.

Currently, this data modelling and view integration is performed manually, but we are developing MetaCASE support for it. An editor is being developed which will support CoCoA diagramming of individual ISMLs. A view integration tool will support drawing (copying while maintaining semantic links) from individual ISML diagrams, specification of synonyms and homonyms between different views, and renaming to account for these view inconsistencies. This tool will also allow further addition of generalisation and specialisation links and super- and sub-classes. The information stored by these tools will be used for (semi-)automatic generation of CASE tool repositories, and will provide the ISML information used by our other MetaCASE tools.

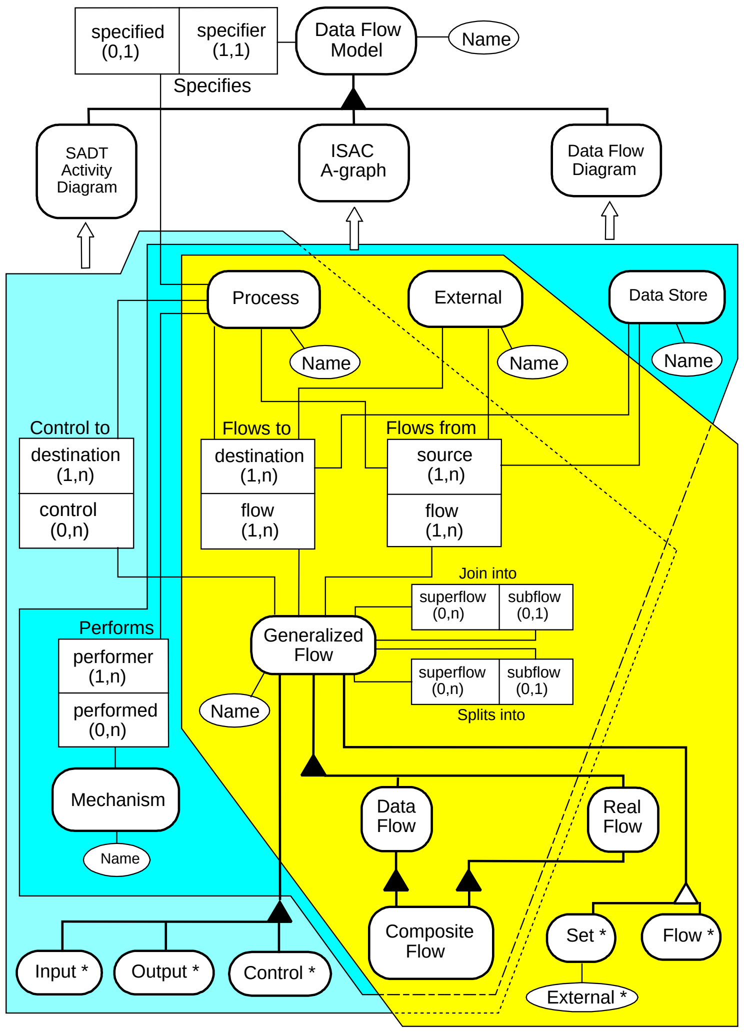


figure 2. Integrated CoCoA views.

4. Inter-ISML Mappings

In the practical operation of an integrated CASE environment, there need to be various links and mappings between the individual CASE tools. The dependencies and links that are identified in the static conceptual data modelling described above must be turned into dynamic actions. These account for the problems which occur when changes made in one tool have an impact on the information which is stored and modelled in another tool.

First, links within the integrated CoCoA model between concepts which belong in different ISMLs must be identified. These static links serve as a basis for the identification of dynamic mappings between ISMLs which must be maintained by the CASE tools.

Next, the nature of the dynamic link needs to be decided. To some extent, the dynamic nature of the links needed to support the dependencies can be inferred using simple heuristics. For example:

1. Concepts which are pure synonyms generally need to have a single base copy of the concept, which should be created, updated, or deleted in each of the individual tools that share that concept. Additionally, the tool user needs to be informed of the change (or reminded if the user is the same one who made the change in another tool).
2. Concepts which are direct or indirect subtypes of concepts in other views may be similarly updated.
3. Concepts which are direct or indirect supertypes of concepts in other views will have to have decisions made about the impact of changes made to them to other notation views.
4. Concepts which have a common supertype with a concept in another view may or may not have an impact on that other view. This needs to be decided by the tool builder.

5. Concepts that are used as a basis for decision-making about other concepts in another view, or are part of an entire view that is used as a basis for decision-making about other concepts, will need to have the impact of changes made to them assessed in the context of that other view.

Heuristics may not identify all dependencies in need of support. Tool integrators need to identify additional dependencies and decide on the dynamic nature of the mappings.

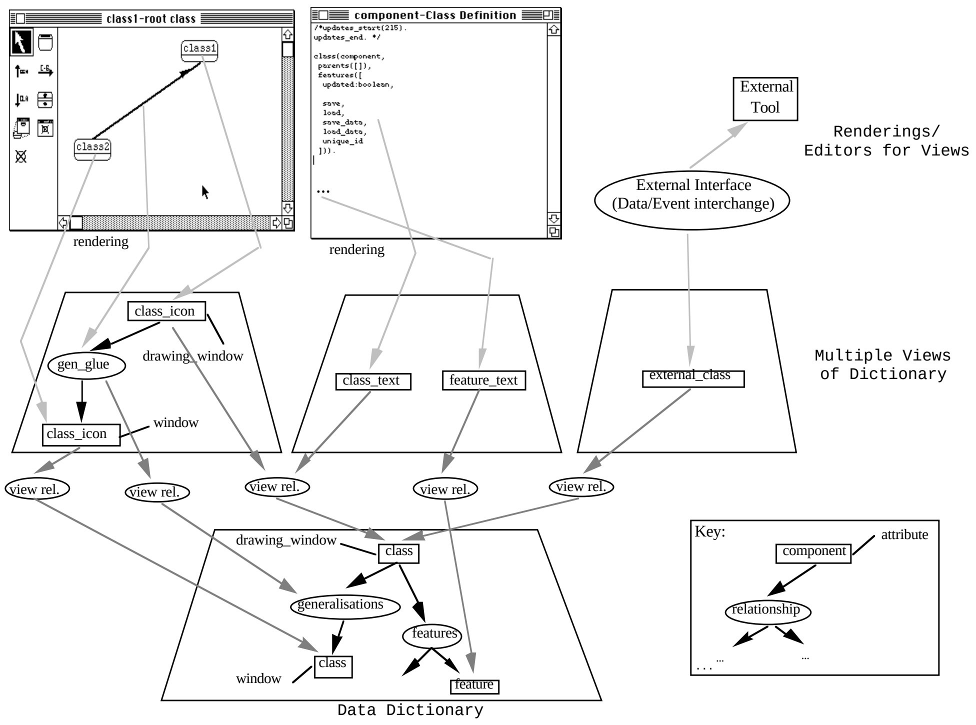
Finally, the dynamic links/mappings are implemented using the link constructs provided within our CASE tool implementation framework, MViews [2, 4]. The constructs provided in the library of MViews links basically reflect the dynamic nature described above.

Currently, identification of links and decision-making about their dynamic nature is accomplished manually. Our MetaCASE environment will support these activities by using data stored by the CoCoA modelling. Integration tools will be used to code the above heuristics, in some cases automatically following decision trees to identify links and decide on an appropriate mappings and dynamic link mechanism. In other cases, the CASE tool builder will make a decision about the appropriate mapping and/or dynamic link mechanism. Once the appropriate mechanism is decided, the environment can generate the appropriate implementation code.

5. Tool Implementation

MViews is an object-oriented framework for constructing new multi-view editing environments [2, 4]. New environments are constructed by specialising framework classes to describe the data dictionary and program representation for ICASE environments. A graph-based structure is used to describe both the software system data and multiple views of this data. Views are rendered and manipulated in concrete textual and graphical forms.

Figure 3 shows an example of an ICASE environment for object-oriented software development, SPE, developed using MViews [5]. The data dictionary describes classes, attributes and methods (collectively called “features”), inter-class relationships, and class and method implementation code. SPE provides integrated graphical OOA/D views and textual implementation and documentation 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, such as Dora [9], FIELD [10], and Software thru Pictures [12].



```
/*updates_start(215).
updates_end. */

class(component,
 parents([]),
 features([
  updated:boolean,

  save,
  load,
  save_data,
  load_data,
  unique_id
 ])).

 ...
```

figure 3. An example ICASE environment built using MViews.

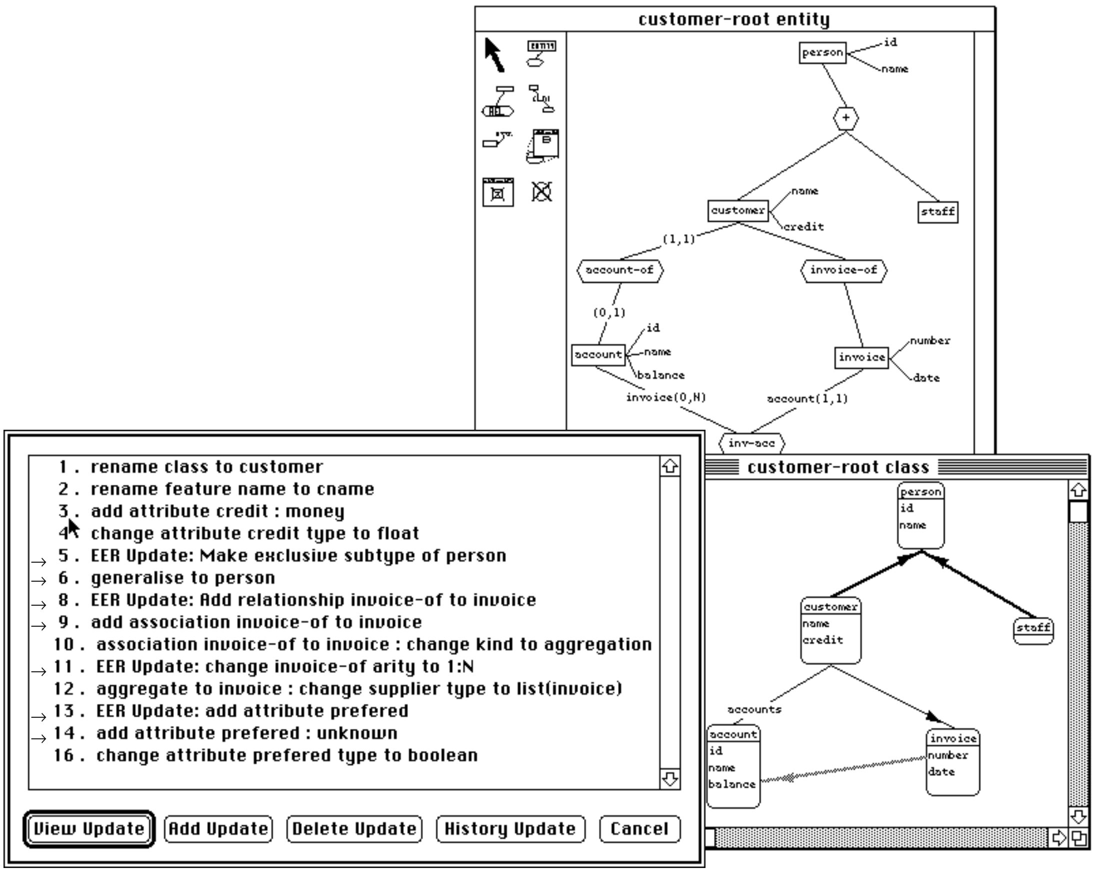
MViews supports very flexible inter-component consistency management by generating, propagating and responding to *change descriptions* whenever a component is modified. A change description documents the exact change in the state of a component and is propagated to all relationships the component participates 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 [8].

In addition to SPE, we have used MViews to build an environment for integrated entity-relationship modelling and relational schema implementation, MViewsER [3], an environment for constructing building designs using the EXPRESS and EXPRESS-G notations [1], and an environment for designing dialog interfaces using integrated textual and graphical views [8].

MViews is implemented in Snart, an object-oriented extension to Prolog. Currently, environment implementers specialise Snart classes to define new environment data dictionaries, multiple views, and view renderings and editors. Persistent objects are used to make data dictionary and view persistency management transparent for ICASE implementers.

Our MetaCASE environment will allow ICASE developers to use the ISML conceptual data models described in section 3 as specifications for generating MViews repositories. It will also allow developers to describe multiple views of these conceptual data models, one for each ISML tool. The user interface for each view will then be specified by designing icon, glue and textual component appearances, and linking these specifications to the underlying view conceptual data models. Specification of user interface behaviour will allow MViews view editors to be generated.

6. Tool Integration



```
    1 .  rename class to customer
    2 .  rename feature name to cname
    3 .  add attribute credit : money
    4    change attribute credit type to float
→   5 .  EER Update: Make exclusive subtype of person
→   6 .  generalise to person
→   8 .  EER Update: Add relationship invoice-of to invoice
→   9 .  add association invoice-of to invoice
   10 .  association invoice-of to invoice : change kind to aggregation
→  11 .  EER Update: change invoice-of arity to 1:N
   12 .  aggregate to invoice : change supplier type to list(invoice)
→  13 .  EER Update: add attribute prefered
→  14 .  add attribute prefered : unknown
   16 .  change attribute prefered type to boolean
```

figure 4. A screen dump from OOEER.

SPE and MViewsER were developed independently using the MViews framework. We integrated these two CASE tools to produce OOEER, an ICASE tool supporting integrated OOA/D and EER design, and object-oriented and relational schema implementation [6].

Figure 4 shows a screen dump from OOEER. The OOA/D views are kept consistent with **all** changes to the EER views, and vice-versa, even when a direct translation is not possible by the environment. The dialog shown holds change descriptions (the “modification history”) for the customer OOA class. The change descriptions highlighted by ‘→’ were actually made to the EER view (diagram) and automatically translated into OOA/D view updates (where possible) by OOEER. Unhighlighted items were made by the designer to the OOA view to fully implement “indirect” translations that could only partially by implemented by OOEER.

SPE and MViewsER were integrated to produce OOEER by defining an integrated data dictionary, illustrated in figure 5, based on an integrated conceptual data model as described in section 3. ISML mappings described in section 4 were used to link the different components and relationships in each notation’s data dictionary. The mappings 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 ISML’s data dictionary. Neither SPE nor MViewsER were modified in any way to support this integration process to produce OOEER.

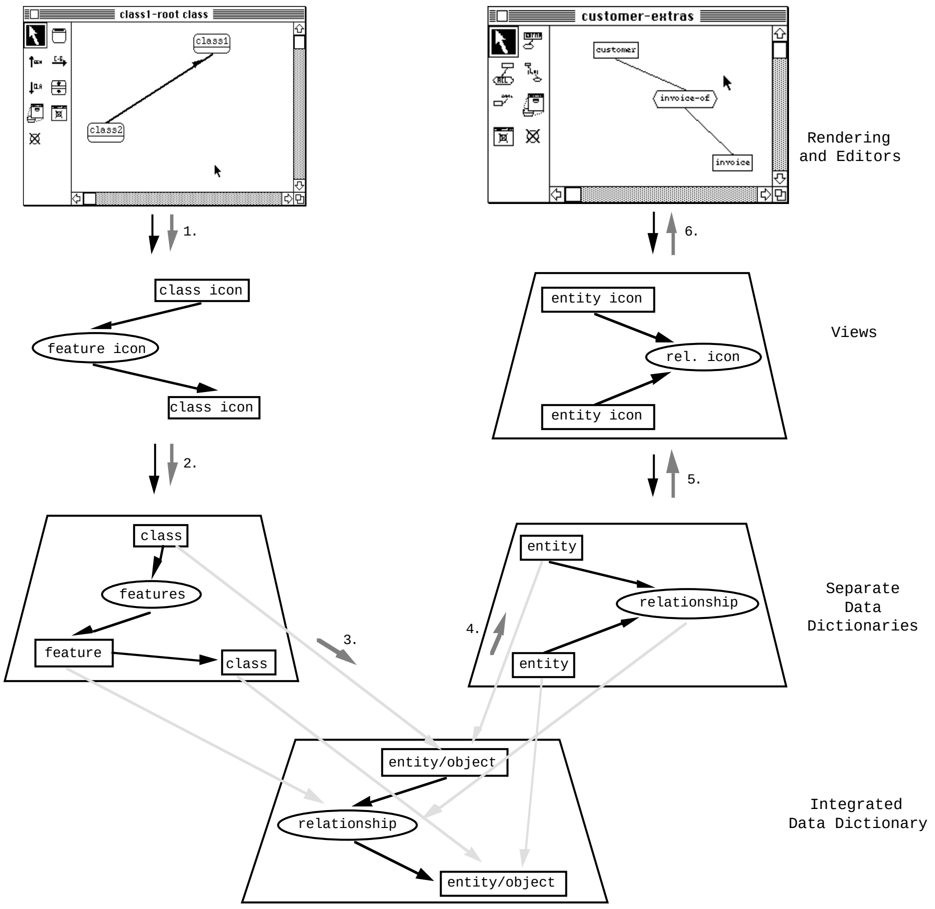


figure 5. Integrating SPE (OOA/D) and MViewsER (EER) ISEEs.

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 when SPE data changes, and respond to these change descriptions to update the integrated data dictionary repository (3). When the integrated data dictionary components change, the inter-repository relationships linking MViewsER’s repository components translate the change descriptions on the integrated dictionary components into updates on the 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).

We have implemented these inter-repository relationships as specialisations of MViews’ generic many-to-many relationships, allowing several components from one repository to be related to several components in another repository. The inter-repository relationships are currently automatically created by OOEER according to the ISML links, as identified in section 4. We are extending OOEER to support user-defined relationships, which will allow designers to link different components and have limited consistency management maintained between them by OOEER. Recent extensions to MViews and SPE to support CSCW facilities [7] allow multiple designers to collaborate using integrated OOEER notations.

We are implementing OOD, state transition diagram and data flow diagram views, to be integrated into OOEER. Currently, new tools are added by performing the following manually:

1. An additional CoCoA model of the view to be supported by a new tool is created, and this view is then merged into the current (so far) integrated CoCoA model.
2. New links then need to be identified, specified, and created between the new tool and the existing tools in the environment.
3. A new tool for the notation is implemented using MViews, including a repository and integrated view editors
4. The new tool repository is integrated with existing tools, currently done by creating a new repository for the new integrated view. This is made into the new root in the tree of repositories, with links and mappings from it to the old integrated repository (the old root) and the new tool's repository. Eventually, the extent and shape of the tree may force rationalisation and redesign of the repository structure to remove excessive levels and long paths between tools.

Our MetaCASE environment will support the integration of such different ISML tools by generating the inter-repository relationships from their conceptual descriptions discussed in section 4. User-defined relationships and CSCW facilities will also be configurable within the MetaCASE environment.

References

1. Amor, R., Augenbroe, G., Hosking, J.G., Rombouts, W., and Grundy, J.C. Directions in modelling environments. to appear in *Automation in Construction* (1995).

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 CS 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. and Hosking, J.G., “Constructing Integrated Software Development Environments with Dependency Graphs,” Working Paper, Department of Computer Science, University of Waikato, 1994.

5. Grundy, J.C., Hosking, J.G., Fenwick, S., and Mugridge, W.B. Chapter 11 in *Visual Object-Oriented Programming*, Burnett, M., Goldberg, A., Lewis, T. Eds, Prentice-Hall (1994).

6. Grundy, J.C. and Venable, J.R. Integrating CASE tools via hierarchical data model integration. to appear in *Proceedings of CAiSE'95*, LNCS, Finland, June 1995, Springer-Verlag.

7. Grundy, J.C., Mugridge, W.B., Hosking, J.G., and Amor, R. Support for Collaborative, Integrated Software Development. to appear in *Proceedings of the 7th Conference on Software Engineering Environments,* IEEE CS Press, Netherlands, April 1995.

8. Grundy, J.C., Hosking, J.G., and Mugridge, W.B., “Supporting flexible consistency management via discrete change description propagation,” Working Paper, Department of Computer Science, University of Waikato, 1995.

9. Ratcliffe, M., Wang, C., Gautier, R.J., and Whittle, B.R. Dora - a structure oriented environment generator. *IEE Software Engineering Journal 7*, 3 (1992), 184-190.

10. Reiss, S.P. Connecting Tools Using Message Passing in the Field Environment. *IEEE Software 7*, 7 (July 1990), 57-66.

11. Venable, J.R. *CoCoA: A Conceptual Data Modelling Approach for Complex Problem Domains,* Ph.D. thesis, State University of New York at Binghampton, 1993.

12. Wasserman, A.I. and Pircher, P.A. A Graphical, Extensible, Integrated Environment for Software Development. *SIGPLAN Notices 22*, 1 (January 1987), 131-142.

13. Wieringa, R.J. Combining static and dynamic modelling methods: a comparison of four methods. to appear in *Computer Journal* (1995).