4

Towards an integrated environment for method engineering

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
In order to facilitate better Information Systems Development (ISD), Method Engineering technqiues and tools are needed that support flexible creation, modification, and reuse of ISD methods and tools for use on specific problem domains. A metamodelling notation is needed for specifying and integrating different design notations. MetaCASE support is required for building, reusing and evolving tools for these design notations. Process modelling tools for both the coordination of these design notation tools and the evolution of software processes are also needed. We describe our work on developing an integrated environment which supports metamodelling, metaCASE and flexible software process modelling, and illustrate its use for supporting Method Engineering.

Keywords
Method engineering, metamodelling, metaCASE, software process modelling

1   INTRODUCTION

Information Systems Development (ISD) methodologies are generally assumed to be situation-independent. However, there are a multitude of different development methods and techniques that each have various advantages and disadvantages, some of which relate to the problem domain or the development context. A stream of research has developed investigating the possibility to choose, tailor, or engineer the development method accordingly. Kumar and Welke (Kumar, 1992) coined the term methodology engineering and postulated this new field, i.e. engineering a new ISD methodology by composing it from various techniques in order to address problems in a particular domain. Vessey and Glass (Vessey, 1994) noted that, in any case, system developers adapt and modify the methods that they use to the situation and their preferences. Recently, Harmsen and Brinkkemper (1995) found that due to the increasing complexity of Information Systems, development teams often require methods tailored to a particular system development situation, which they term Situational Method Engineering.

Developers may need to create a new method from scratch, modify (e.g. incrementally improve or tailor) an existing method, or reuse parts of various methods and techniques and recombine them into a new method, or any combination of the above. Developers may even need to modify and adapt the development method while the development process is ongoing. Our goal is to support this flexible sort of situational method engineering. In order to facilitate this, developers need flexible support for:

•   Design notation metamodelling and notation integration. This allows developers to specify data models for the design notations they wish to use for development of a system, and in the case of multiple design notations, to specify common information that will be shared between the notations. Developers should be able to reuse all or parts of existing notations, and be able to integrate different notations when one notation best supports modelling part of the problem domain, and another notation is better suited to another.
•   Tool construction facilities. These are used to build or modify CASE tools to support the various design notations to be used. This includes the ability to keep information shared by different design notations consistent i.e. to keep different notation repository information consistent under change. Developers also need to specify the editors and rendering of notation data models they desire.
•   Software process modelling and work coordination. Process modelling specifies which notations and tools will be used for different aspects of the system under development. Work coordination support is needed to coordinate tool usage. Evolution and reuse of process models allows developers to improve their development processes from one project to another. Modelling the Method Engineering process itself provides a meta-process level which helps to improve Method Engineering on subsequent projects.

Ideally a Method Engineering environment should support all of these activities in an integrated fashion. Developers should be able to define and/or reuse software processes either for developing a new system or for modifying an existing system and its existing descriptions. They should be able to tailor existing design notations in either case. CASE tools supporting the required notations should be built using the notation metamodels as repository specifications, and common information in different tool repositories should be kept consistent. Developers should be able to flexibly define and revise software processes during system development, and be able to reuse these models on new projects.

   Our approach is to combine techniques and tools from three distinct, yet related, areas of our recent research. We have developed the CoCoA meta-modelling notation (Venable 1993, Venable 1995) and have used this for design notation metamodelling and integration (Grundy, 1995a, Grundy, 1995b, Venable, 1995). We have developed the MViews framework for constructing CASE tools and integrated Information Systems Engineering Environments (ISEEs) (Grundy, 1993, Grundy, 1995a) and used this to develop ISEEs which support multiple design notations [Grundy95a, Grundy95b]. Recently we have been developing a tool for the coordination of work in CSCW systems (Grundy, 1995c), which also supports flexible software process modelling. This paper describes our current work developing an integrated environment for the definition, construction and coordination of ISEEs using these techniques.

2   RELATED RESEARCH

Current approaches to notation integration, CASE and metaCASE, and method engineering support tools, go some way to addressing the Method Engineering aims from Section 1, but do not completely satisfy them. Some work has been done on the static integration of notations. Venable (1993) has performed detailed analyses and integrations of both data flow models and conceptual data models. Campbell and Halpin (1994) have analysed levels of abstraction for conceptual schemas. Falkenberg and Oei (1994) have proposed a metamodel hierarchy.

Wieringa (1995) has compared JSD, ER modelling and DFD modelling. Data modelling has been used to compare different notations (Nuseibeh, 1992) and support methodology engineering (Heym, 1992). Process-modelling has also been applied to compare and integrate notations (Song, 1992).

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 (Wasserman, 1987, Reiss, 1990). These ICASE environments allow developers to analyse and design software using a variety of different notations, with limited inter-notation consistency. Such tools do not generally support complex mappings between the design notations, such as propagating an ER relationship addition to a corresponding OOA/D or NIAM diagram. As an example, Software thru Pictures™ (Wasserman, 1987) uses a single metamodel repository for all notation diagrams, although it only supports basic forms of internotation consistency. 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 (Meyers, 1991). For example, MethodMaker from Mark V Systems (Mark, 1995a) allows new notations and methods to be built, but provides very limited inter-notation consistency management facilities. FIELD (Reiss, 1990) and Dora (Ratcliffe, 1992) provide abstractions for keeping multiple tools and textual and graphical views consistent under change. They do not, however, provide any mechanism for propagating changes between views which can not be directly applied by the environment, such as ER relationship changes to NIAM or OOA/D relationship changes. Thus changes which can not be automatically translated to another notation are not supported.

Process-centred environments utilise information about software processes to enforce or guide development. Examples include Marvel (Barghouti, 1992), CPCE (Lonchamp, 1995), and ConversationBuilder (Kaplan, 1992). These environments usually provide low-level text-based descriptions of work rationale, and often do not effectively handle restructuring of development processes while in use (Swenson, 1993). ProcessMaker (Mark, 1995b) supports the definition and use of multiple process diagrams, but only supports limited integration and no event handling for I-CASE tools. Computer-Aided Method Engineering (CAME) tools, such as Decamerone (Harmsen, 1995) and Method Base (Saeki, 1993), provide support for configuring development processes and tools to a particular application, but often utilise complex textual specifications, and don't facilitate coordination of different notation tools during development.

3   THE COCOA META-MODELLING LANGUAGE

3.1   CoCoA

We have been using the CoCoA conceptual data modeling language (Venable, 1993) as a meta-model for modelling Information System Modelling Languages (ISMLs). CoCoA is designed to support modelling of complex problem domains and extends existing Entity Relationship (ER) models. Figure 1 depicts the seven main CoCoA abstractions. Entities are the things in a problem domain and attributes describe and/or identify them (Figure 1 (a)). Named relationships have the semantics of ER relationships, and are composed of named roles, played by entities. Cardinality constraints are indicated with each role (Figure 1 (b)). CoCoA supports generalization and specialization, and where specialization is based on a partitioning attribute, that attribute is shown (Figure 1 (c)). CoCoA extends other ER models by the implicit use of categories, allowing the entity planing a role in a named elationship to be one of one or more enity types, shown by connecting more than one entity (type) to the same role (Figure 1 (d)). CoCoA derives its name from a fifth data modelling concept, that of <u>Co</u>mplex <u>Co</u>vering

Aggregation. Covering aggregation distinguishes the aggregation of entities into composite entities from the aggregation of attributes into entities. Complex covering aggregation is distinguished from simple covering aggregation in that aggregation of named relationships into the composite entity is allowed (Figure 1 (e)). CoCoA supports aliases, which are useful for model integration, showing old local names together with standardized names for synonyms (Figure 1 (f)). Derived concepts (attributes, entities, named relationships, or covering aggregation relationships) are annotated with a '*' (Figure 1 (g)).
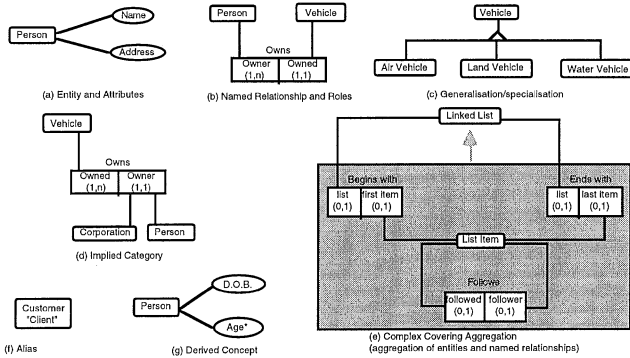

Figure 1   The CoCoA model notation.
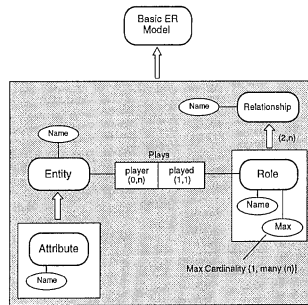
3.2   MetaModelling with CoCoA

Figure 2   Metamodel of core ER concepts.

We have used CoCoA to derive conceptual data models for the ER, NIAM, DFD, STD and OOA/D design notations. As an example, the data model describing the fundamental abstractions of ER models is shown in Figure 2. Enities are named and have zero or more named attributes. Relationships are named and have two or more named roles. Roles link entities and relationships and may include a maximum cardinality. Extensions to this basic ER schema include provision for entity subtyping, optional and mandatory roles, and distinguished key attributes of entities (Venable, 1993).

Figure 3 shows NIAM's main abstractions. A NIAM entity is named and may have a reference, made by one or more named labels. Fact types are named and have one or more roles. The "derived" attribute of the fact type entity is marked as derived (by the asterisk) because its value is true if it is related to a derivation rule. Roles link entities to facts, and are named. Nested fact types are both entities and facts, i.e. they have roles but also behave as entities, being linked to zero or more facts via further roles. A CoCoA model of other NIAM constraints is omitted for brevity, but can be found in (Venable, 1993). NIAM derivation rules are not specified further because they are not fully specified by Nijssen and Halpin (1989). Other notation meta-models can be found in (Venable, 1993, Grundy, 1995a).
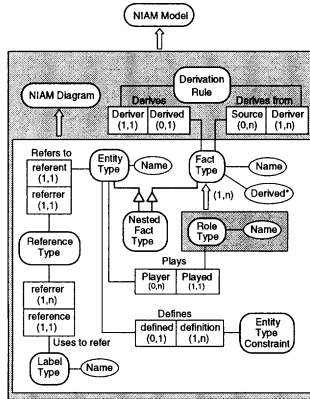

Figure 3   Metamodel of core NIAM concepts.

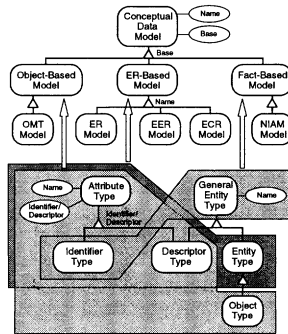## 3.3   Notation MetaModel Integration with CoCoA

Figure 4   An integrated conceptual data model.

We have developed integrated data models which capture the overlaps between ER, EER, OMT's object model, and NIAM. Figure 4 shows a *partial* metamodel integrating the entity and attribute data modelling aspects of ER, EER, NIAM, and OMT. The ER and OMT models differentiate between entities and attributes, whereas NIAM integrates these concepts into a general entity type. The main difference between the OMT and ER conceptual data models is OMT's support for class methods. The overlaps between the notations are indicated by covering aggregation showing the composition of each data model from the integrated data model entities and relationships. Further discussion of these and of relationship type classifications is in (Venable, 1993).

4   THE MVIEWS FRAMEWORK

4.1   MViews

Our design notation environments are implemented as a collection of Snart classes, specialised from the MViews framework (Grundy, 1993). MViews supports the construction of new ISEEs by providing a general model for defining software system data structures and tool views, with a flexible mechanism for propagating changes between software components, views and distinct software development tools. Figure 5 shows an example of the structure of SPE, an ISEE for object-oriented software development. ISEE data is described as *components* with *attributes*, linked by a variety of *relationships*. Multiple views are supported by representing each view as a graph linked to the base software system graph structure. Each view is rendered and edited in either a graphical or textual form. Distinct environment tools can be interfaced at the view level (as editors), via external view translators, or multiple base layers may be connected via inter-view relationships, as described in (Grundy, 1994).

   When a software or view component is updated, a change description is generated. This is of the form UpdateKind(UpdatedComponent, ...UpdateKind-specific Values...). For example, an attribute update on Comp1 of attribute Name is represented as: update(Comp1,Name,OldValue,NewValue). All basic graph editing operations generate change descriptions and pass them to the propagation system. Change descriptions are propagated to all related components that are dependent upon the updated component's state. Dependents interpret these change descriptions and possibly modify their own state, producing further change descriptions. This change description mechanism supports a diverse range of software development environment facilities, including semantic attribute recalculation, multiple views of a component, flexible, bi-directional textual and graphical view consistency management, a generic undo/redo mechanism, and component "modification history" information (Grundy, 1995d). New environments are constructed by reusing abstractions provided by an object-oriented framework, and ISEE developers specialise MViews classes to define software components, views and editing tools. A persistent object store is used to store component and view data.

   MViews environments support version control and collaborative facilities via the C-MViews extensions to MViews (Grundy, 1995d). Version revision, alternates and merging are supported by having change descriptions cached in a number of version records for components and views. Merging of alternate versions is carried out by sucessively reapplying one alternate's change descriptions to the other alternate component. Any merge conflicts (structural or semantic) are presented to the merging user. Semi-synchronous and synchronous editors are provided for views by propagating change descriptions on a view to other users' environments as they occur. With semi-synchronous editing, these change descriptions are presented to collaborating users, who may then choose to incorporate them into their own view alternatives. For synchronous editing, a central server "owns" the shared view, and all edits must be sent to this server for actioning and propagation to other users. Fine-grained view component locking

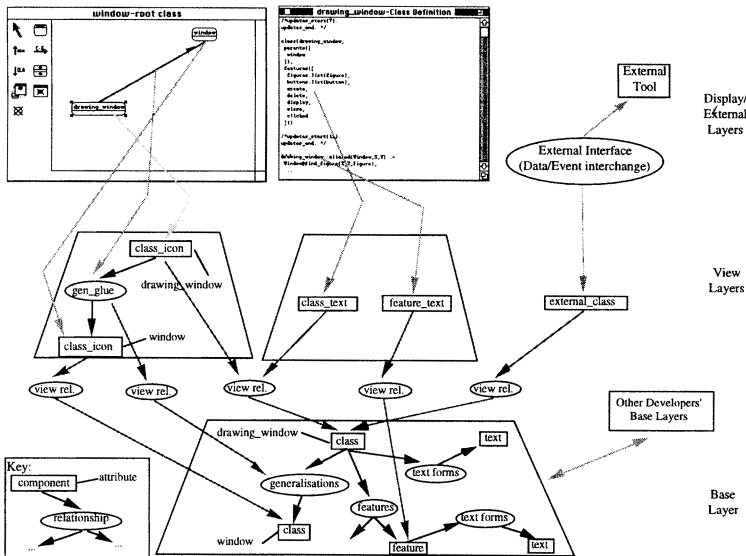is maintained by the server to ensure no simultaneous component update is permitted by multiple users.

Figure 5   The MViews Architecture.

4.2   Notation Integration with MViews

In addition to SPE, we have developed several other ISEEs using MViews. MViewsER provides integrated Entity-Relationship diagrams and textual relational schema. MViewsER has been integrated with SPE to produce OOEER, an integrated environment for OOA/D and EER modelling (Grundy, 1995a). MViewsNIAM provides NIAM modelling views, and has been inegrated with MViewsER to produce NIAMER (Venable, 1995). MViewsDP provides a graphical drag-and-drop interface builder for dialog boxes, with the dialog interface and validation rules being defined in textual views (Grundy, 1995d). EPE is an environment for constructing EXPRESS specifications and corresponding EXPRESS-G diagrams (Amor, 1995a). C-SPE and C-MViews provide collaborative, integrated software development support via synchronous, semi-synchronous and asynchronous editing (Grundy, 1995e).

Figure 6 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.
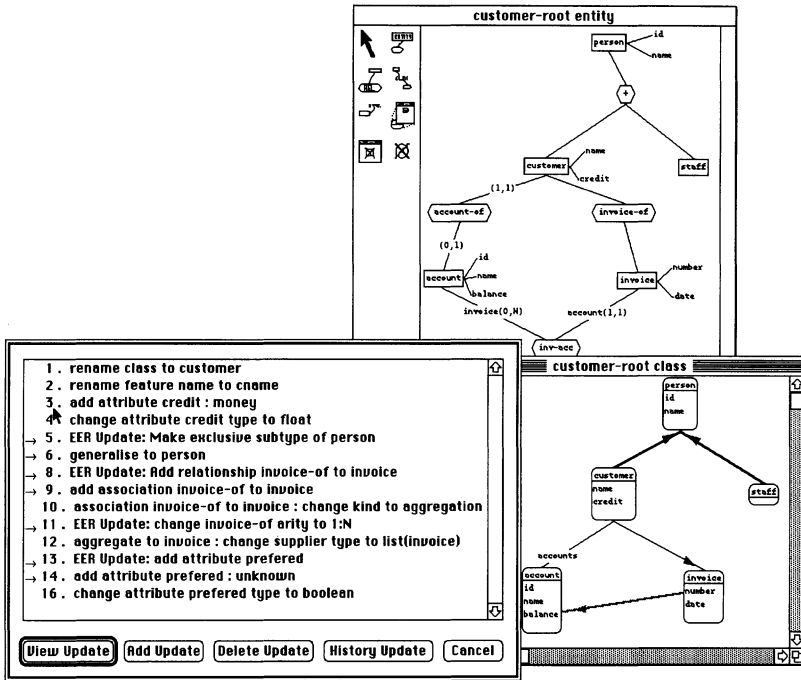
Figure 6   Integrated OOA/D and EER views in OOEER with bi-directional consistency.

The OOEER integration was achieved by adding an additional data dictionary graph level below the data dictionaries of the SPE and MViewsER tools. This layer is responsible for translating, where possible, between the different notations and notifying tools where automatic translations are not possible. Neither SPE nor MViewsER required any significant change to achieve this integration. Figure 7 shows an example of the structure of OOEER. Figure 7 illustrates this integration process. 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).

5   THE SERENDIPITY PROCESS MODELLING TOOL

ISEEs should support the coordination of cooperative work activities that is inherent within ISD (Krant, 1995). Therefore, CSCW features are needed in ISEEs. An ISEE should support users in collaboratively planning and executing work activities, as well as in being informed about

and maintaining their awareness of relevant work by others, the contexts in which those other users' work is carried out, and the rationale for the decisions they have made. In particular, support is needed for defining activities to be done (plans), coordinating the planning activity itself (meta-plans), and restructuring the history of work done to more effectively convey intent ("rewriting history"). Unfortunately most existing workflow systems are inadequate for real-world applications due to many exceptions to the workflows and their inability to adapt to changing work processes (Swenson, 1993). Similarly, must existing process modelling tools utilise either complex, textual specifications which are inaccessible to many end-users, or do not support facilities for integration and event handling with existing tools.

We have developed Serendipity, a process modelling, enactment and work planning environment, which also supports flexible event handling mechanisms, group communication, and group awareness facilities (Grundy, 1996). Fig. 8 shows a Serendipity process model for updating a software system ("m1:modify system-process"). The notation is an adaptation and extension of Swenson's Visual Planning Language (Swenson, 1993), which does not support artefact, tool or role modelling, nor arbitrary event handling mechanisms.
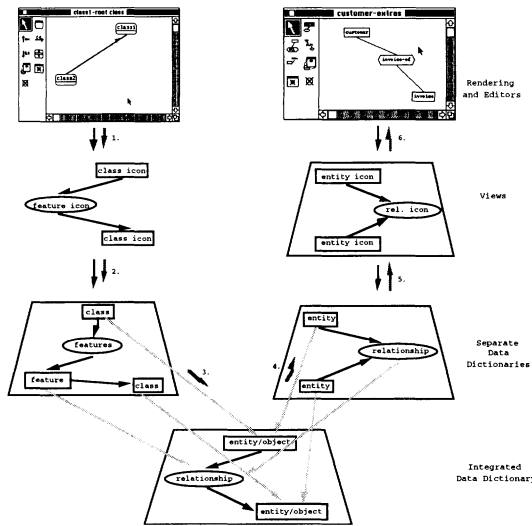

Figure 7   Integrating SPE and MViewsER using an integrated data model.

Stages describe steps in the process of modifying a software system, with each stage containing a *unique id*, the *role* which will carry out the stage, and the *name* of the stage. Enactment *event flows* link stages. If labelled, the label is the *finishing state* of the stage the flow is from (e.g. "finished design"). The shadowing of the "m1.2:implement changes" stage indicates that multiple implementers can work on this stage (i.e. the stage has multiple subprocess enactments). Other items include *start stages*, *finish stages*, *AND* stages, and *OR* stages (empty round circle). Underlined stage IDs/roles mark presence of a *subprocess* model, for example "m1.1:plan changes-subprocess" is a subprocess for "m1.1:design changes". The italicised "check out design" stages in this subprocess model indicate stages reused from a *template* process model.
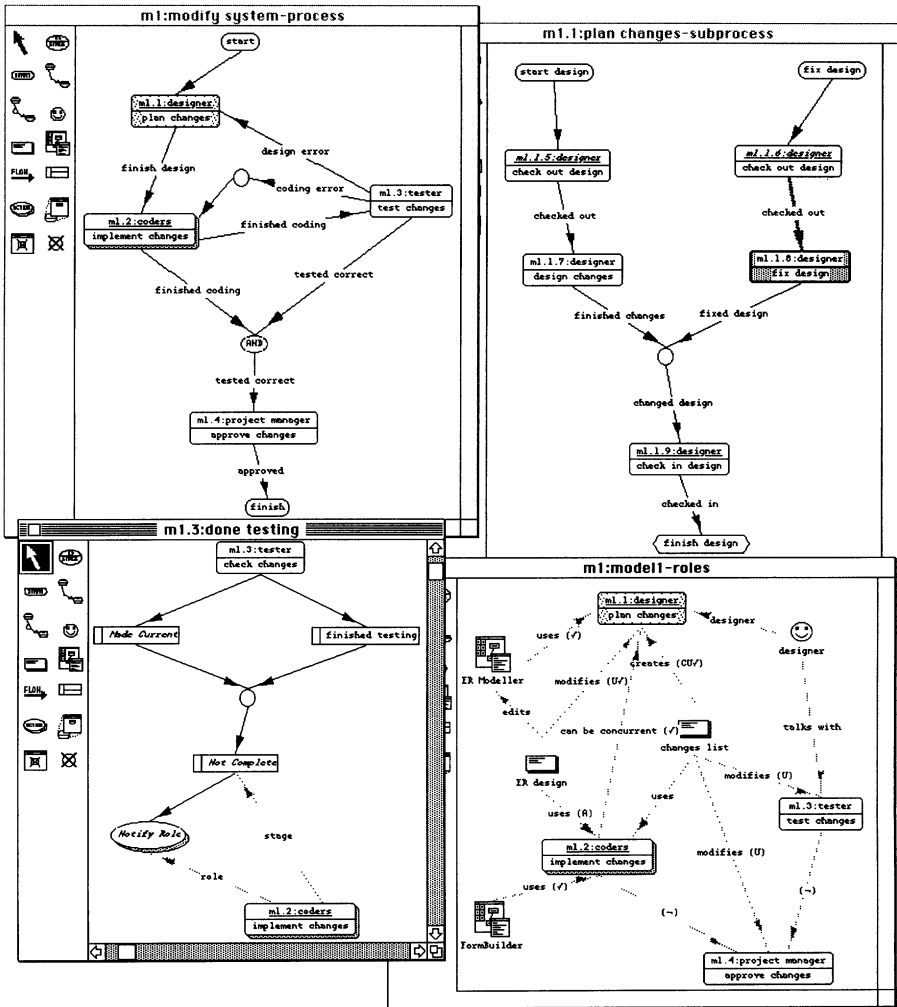
Figure 8   Software process model views and dialogs in Serendipity.

Serendipity supports artefact, tool and role modelling for processes, as in "m1:model1-roles", which shows a different perspective of "m1:model1-process". *Usage connections* indicate how stages, artefacts, tools and roles are used. Optional annotations indicate: whether data is created (C), accessed (A), updated (U), or deleted (D); whether a stage must use only the tools, artefacts or roles defined (√); and whether a stage cannot use a particular tool, artefact or role (¬). If a stage is linked to another stage by a usage flow, "√" specifies the stage may be enacted

when the other stage is enacted, while "¬" specifies the stages can not be enacted at the same time.

In addition to specifying the static usages and enactment event flows between process model stages, Serendipity supports *filters* and *actions*, which process arbitrary enactment and work artefact modification events. View "m1.3:done testing" shows an example of enactment event filtering. The filters "Made Current" and "finished testing" determine if "m1.3:check changes" has been made the current enacted stage or has been finished. If so, then if the "m1.3:implement changes" process has not completed (determined by filter "Not Complete"), the role associated with this stage is notified of testing being started or completed.

Stages are *enacted* for a project, highlighted by colour and shading, as shown in Figure 8. The shaded stage with a bold border ("m1.1.8:fix design") is the *current enacted stage* for the user i.e. their current work context. As a stage completes in a given finishing state, event flows with this state name (or no name) activate to enact linked stages. Enactments of stages are recorded, as are process model changes, and all enacted stages for a user can be shown in a "to-do" list dialog.

6   AN INTEGRATED METHOD ENGINEERING APPROACH
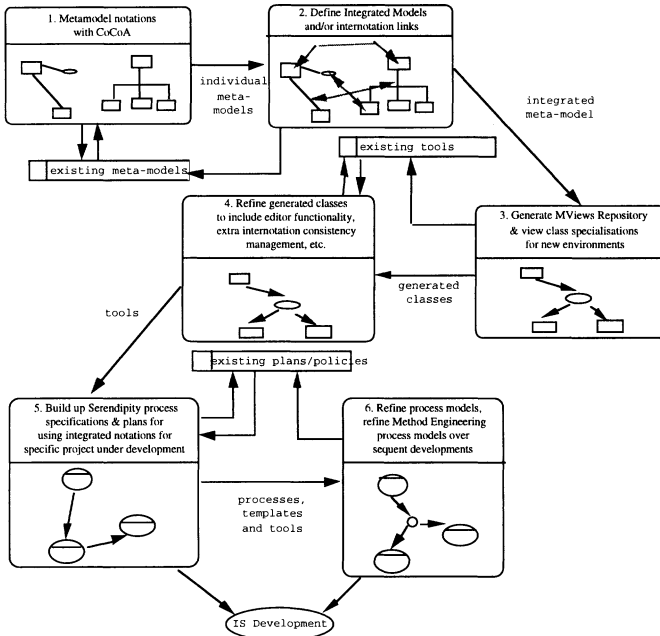

Figure 9   The method engineering process with our integrated tools.

We are currently building an MViews-based environment for CoCoA modelling, which will form the basis of an integrated environment for Method Engineering with our tools. Figure 9 shows how this tool will be used to generate MViews framework classes for specifying new or

modified tool repositories and views. Developers will augment these specifications with appropriate editor configurations and notation renderings, and any additional consistency management techniques not generated from the CoCoA metamodels. As our Serendipity work coordination tool can be used with any MViews environments, developers will then be able to specify appropriate plans (i.e. process models) for different systems under construction. This may include enabling usage of certain tools and artefacts for certain parts of the new system development or to particular groups of developers. Once these plans have been created, developers can later abstract these plans to form policies and reuse their policy process models for subsequent systems, and thus incrementally refine these process models. CoCoA models and MViews environments and tools can also be created and/or modified from one project to the next to build up appropriate tools for each system development.

5.1   CoCoA Metamodelling of Notations

The first step (#1 in Figure 9) is to build up CoCoA metamodels of the desired design notations to be used on a new system development. This might include the reuse of previous CoCoA models, the combination of parts of one metamodel with another, or the development of new metamodels which are problem-specific. In previous metamodelling with CoCoA, we have used a drawing editor to produce these metamodels (Grundy, 1995a, Venable, 1995). We are currently implementing an MViews tool for CoCoA modelling which will be used to construct new CoCoA models, and will include multiple views of CoCoA models and libraries of views and models to assist in model reuse.

   As an example, we have recently integrated our NIAMER (supporting NIAM and ER views) and OOEER (supporting OOA and EER views) environments with the MViewsDP form/report designer by hand. Using our CoCoA modelling tool instead, integration of these tools would initially begin using our integrated CoCoA/MViews environment to metamodel each notation that is to be used on a development project.

5.2   Conceptual Notation Integration with CoCoA

Integrating different design notations with CoCoA involves either the definition of integrated models or specifying links between components of one model and related components in another model (Grundy, 1995a, Venable, 1995). In addition, dynamic mappings must be specified between these notation components i.e. what happens to related components when a component instance is changed. For example, in OOEER, if an ER relationship is added between two entities, a default association relationship is added between the corresponding two object/classes in the OOA model (Grundy, 1995a). Our MViews editor for CoCoA will support both the static integration and/or linking of notation components, and the specification of dynamic mappings between notation components. Static integration is a straightforward view integration, supported by aspects of the CoCoA data modelling language. We are currently adapting a view mapping language (Amor, 1995b) which will allow us to declaratively specify the dynamic notation mappings in this tool. In previous notation integration we have informally specified these dynamic mappings using English and informal diagrams, but this is not sufficient to generate internotation relationships for MViews tool integration.

   In our integrated ISEE, integration of OOEER, NIAMER, and MViewsDP was implemented by adding inter-repository relationships between repositories, in addition to the hierarchial repository relationships used in NIAMER and OOEER. Appropriate links were specified between the components of the CoCoA metamodels for each notation and then dynamic mappings were defined between related components. In future environment integration, this will all be carried our within our CoCoA modelling tool.

5.3 MViews Tool Generation from CoCoA Models

CoCoA models have been used as the specifications for MViews tool repositories in our previous notation integration work (Grundy, 1995a, Grundy, 1995b, Venable, 1995). However, MViews repository and view information has been hand-generated from these models. We are extending our CoCoA modelling tool to generate class interfaces and method code directly from different notation and integrated notation metamodels. Quite a large amount of MViews framework code can be generated in this way: previous development of MViews environments has shown over 60% of the code relates to defining class structures which represent repository and view data, and method code to link these data items in appropriate ways. All of this code can be generated from CoCoA metamodels by our modelling environment. Internotation relationships and a large amount of consistency management code can also be generated in this way, from the static integration and the specification of dynamic mappings in our CoCoA editor, even in some cases the user interface.

For example, if two constructs in two notations are represented by the same entity type in the integrated CoCoA model, a change to one of them in one view results in the same change in the second view/notation. In this case, the user of the second view need only be notified of the change and the repositories updated. Similarly, if the construct changed in one view is a subtype of a construct in another view, changing an instance of the first construct will require the same change on the supertype construct (as long as it has the required attributes). In either of these cases, code for handling the propagation of these changes (including the receiving view's user interface) can be generated automatically. However, the reverse is not true (i.e. we cannot generate the code for propagation of a change to a construct that is the supertype of a construct in another view which needs to be updated).

5.4 MViews Tool Refinement and Integration

While repository and view structures (and some semantic values) can be generated directly from CoCoA metamodels, extra code needs to be written by developers to appropriately configure editors and specify some consistency management code which can not be automaticlly generated. We are allowing developers to further specialise classes generated from our CoCoA modelling tool to define editing mechanisms and internotation consistency management code which can not be specified in a declarative way. For example, developers will specify default techniques for keeping data in different notations consistent declaratively, but may then want to define complex consistency management techniques operationally (i.e. using MViews code). An example from NIAMER is when a NIAM entity is added. Since, in the integrated CoCoA model, a NIAM entity is a supertype of both the EER attribute and entity, it could be mapped to either. NIAMER defaults the automatic translation to adding an entity and allows users of the integrated environment to modify the ER entity to an attribute if desired. This default code could also be generated, but might be incorrect if the default should have been to add an attribute instead. In that case, a developer would need to rewrite a small amount of MViews code. Alternatively, other user interfaces might be desired, such as preseting the user with a menu to add either an entity or an attribute to the EER view, adding reconciliation of the change to a to-do list, or simply suggesting a change. We are currently looking at ways to declaratively specify the desired behaviour with annotations to the CoCoA models. Our approach allowing developers to make these alterations by further specialising the generated classes, which avoids the problem of when the CoCoA models are modified and classes regenerated. The further specialised classes are not lost when this regeneration occurs.

Of course, we also need to code the rendering of the notation on the user's screen. This is currently done by hand, but within the MViews library framework. We are looking at how this might be done declaratively with annotations to the CoCoA metamodel, then generating the rendering code.

5.5   Process Model Specification

After building appropriately integrated tools, developers can then specify how these tools are to be used on the particular system under construction. Serendipity allows developers to specify which tools and work artefacts are used for different plan stages and hence which tools/artefacts can be used for a particular software process. Serendipity process models can guide developers i.e. suggest which tools are appropriate for different development tasks. They can also be used to suggest or to enforce the use of specific tools, so, for example, a project manager may specify one development group uses OOA/D modelling while another uses ER/DFD modelling. As these tools have integrated repositories (via OOEER), the designs produced by each group are still integrated and kept consistent.

   In a collaborative, integrated ISEE, users must be informed of changes to work and plan artefacts that are relevant to them and they are currently interested in (Grundy, 1995c). Some changes a developer makes are directly relevant to their collaborators, such as renaming or deleting entities and attributes, and collaborators should be informed of these immediately. Other changes, such as the addition of new entities, relationships, attributes or forms and reports can be sent for later perusal, as they have more limited effects on collaborators' work. Low-level changes, such as the implementation of procedures, forms or reports not affecting a collaborator's work need not be presented. Collaborators can see from plan histories and various active stages the kinds of activities another developer is doing, and may choose to view these changes or modified artefacts on-demand, using any of the informing mechanisms described above.

   In most CSCW environments, only artefact-level information about changes is presented to collaborators, either directly updating their work artefact views or using version control facilities to indicate changes made by other users. Serendipity provides collaborating users not only with change descriptions describing actual work (or plan) artefact changes, but also with extra information about the work context in which the changes were carried out. Examples of this work coordination can be found in (Grundy, 1995c, Grundy, 1996).

5.6   Tool/Process Refinement and Reuse

Serendipity views assist in Situational Method Engineering (Harmsen, 1994) by allowing developers to incrementally refine their development methodology, processes and work plans. As process stages record information about the tools to use, artefacts to modify/produce, subsequent stages, and also may be exploded into more detailed plans, they facilitate the engineering of software processes in a manner similar to Method Engineering tools. Our approach has some advantages over comparable notations, such as MEL (Harmsen, 1995), in that its visual nature is more accessible to developers for visualising and modifying plans than the textual notations of other approaches. As Serendipity models were designed for general work process modelling, its high-level nature allows developers to more readily understand and modify process descriptions than text-based process-centred environments or method-engineering tools. It also allows users to modify their process and work plans while a model is in use. Finally, Serendipity allows users to restructure copies of processes and plan histories after completion so that new, improved process model templates can be developed for later reuse.

   Our integrated, collaborative ISEE supports collaborative planning via collaborative editors for Serendipity views, and allowing other Serendipity views to act as meta-process views. Collaborating developers share software process views and can collaborate on modifying these models. The use of these shared models for work context capture and presentation, and specifying interest in changes, allows Serendipity to be used for work coordination, collaborative planning, recording development histories, and method engineering. Figure 10 shows an example of process improvement with Serendipity. The process model is extended to include a "m1.5:check design" stage, to be carried out before coding starts. Coder "john"'s

work plan is also extended by adding 'm1.2.11:modify branch table'. This handles an exception to the work plan due to the addition of the "address" table. Such changes could be made before, during or after the model and plan are used.
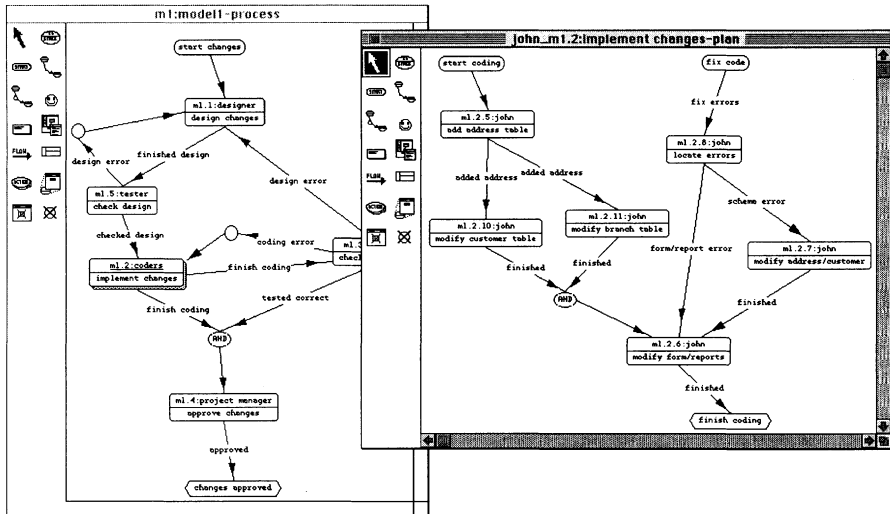

Figure 10   Process and plan improvement example.

Our integrated ISEE allows different tools to be used on the same problem domain, with tool data being kept consistent under change and the tools sharing a consistent user interface. Serendipity allows software processes to be reconfigured during development to better suit a particular development project. Software process models thus evolved can be reused in subsequent development by saving them as reusable templates. The specification of artefacts, roles, CASE tools and interest obligations for plan stages gives our integrated environment similar method engineering capabilities to method engineering tools. In addition, it supports work coordination. Our CoCoA/MViews environment can itself make use of Serendipity views to model, plan and coordinate the Method Engineering process itself. This allows the Method Engineering process to be refined over several projects, in addition to the refinement of the integrated design notation tools.

7   SUMMARY AND FUTURE RESEARCH

We have described our recent work on developing a metamodelling language, CoCoA, notation integration using CoCoA, the construction of integrated Information Systems Engineering Environments based on CoCoA metamodels using MViews, and the development of a work coordination and software process modelling tool, using an extended form of the Visual Planning Language. Used in conjunction, these tools allow system developers to model and integrated different design notations and to construct integrated tools and environments supporting these notations. Developers can plan and coordinate the use of different tools within this environment using our Serendipity tool.

We are currently implementing an MViews environment for CoCoA which will support notation metamodelling and notation integration. MViews classes to implement an integrated environment will be generated from these metamodels, together with internotation relationships and consistency management support. Serendipity will be used to coordinate the use of these integrated environments for different system developments, and will be used by the CoCoA/MViews environment itself to plan, coordinate and refine the Method Engineering process itself. We are also engaged in further research to enhance and add to the CSCW features of our environments and to consider ways to utilise declarative annotations to the CoCoA models to further improve the CoCoA/MViews environment's code generation capabilities.

REFERENCES

Amor, R., Augenbroe, G., Hosking, J.G., Rombouts, W., and Grundy, J.C. (1995) Directions in modelling environments, *Automation in Construction*, **4**, 173–187.

Amor, R.W. and Hosking, J.G. (1995) Mappings: the glue in an integrated system, in *1st European Conference on product and process modelling in the building industryindustry,* A.A. Balkema Publishers, Rotterdam, The Netherlands.

Barghouti, N.S. (1992) Supporting Cooperation in the Marvel Process–Centred SDE, in *Proceedings of the 1992 ACM Symposium on Software Development Environments,* ACM Press, pp. 21–31.

Campbell, L. and Halpin, T. (1994) Abstraction Techniques for Conceptual Schemas, in *Proceedings of the 5th Australasian Database Conference,* Global Publications Services, Christchurch, New Zealand, 17–18 January 1994, pp. 374–388.

Falkenberg, E.D. and Oei, J.L.H. (1994) Meta Model Hierarchies from an Object–Role Modelling Perspective, in *First International Conference on Object–Role Modelling* (ed. Halpin, T. and Meersman, R.), Key Centre for Software Technology, The University of Queensland, Brisbane, Australia, 4–6 July 1994, pp. 310–323.

Grundy, J.C. and Hosking, J.G. (1993) A framework for building visusal programming environments, in *Proceedings of the 1993 IEEE Symposium on Visual Languages,* IEEE Computer Society Press, pp. 220–224.

Grundy, J.C. and Hosking (1994) J.G., *Constructing Integrated Software Development Environments with Dependency Graphs,* Working Paper, Department of Computer Science, University of Waikato.

Grundy, J.C. and Venable, J.R. (1995a) Providing Integrated Support for Multiple Development Notations, in *Proceedings of CAiSE'95,* Finland, June 1995, Lecture Notes in Computer Science 932, Springer–Verlag, pp. 255–268.

Grundy, J.C., and Venable, J.R. (1995b) Developing CASE tools that support integrated design notations, in *Proceedings of the 6th European Workshop on Next Generation of CASE Tools*, pp. 109–116.

Grundy, J.C., Mugridge, W.B., Hosking, J.G., and Apperley, M.D. (1995c) Coordinating, capturing and presenting work contexts in CSCW systems, in *Proceedings of OZCHI'95,* Wollongong, Australia, Nov 28–30 1995, pp. 146–151.

Grundy, J.C., Hosking, J.G., and Mugridge, W.B. (1995d) Supporting flexible consistency management via discrete change description propagation, to appear in *Software – Practice and Experience.*

Grundy, J.C., Mugridge, W.B., Hosking, J.G., and Amor, R. (1995e) Support for Collaborative, Integrated Software Development, in *Proceeding of the 7th Conference on Software Engineering Environments,* IEEE CS Press, Netherlands, April 5–7 1995, pp. 84–94.

Grundy, J.C. (1996) *Serendipity: integrated environment support for process modelling, enactment and improvement*, Working Paper, Department of Computer Science, University of Waikato.

Harmsen, F., Brinkkemper, S., and Oei, H. (1994) Situational Method Engineering for Information System Projects, in *Proceedings of the IFIP WG8.1 Working Conference CRIS'94* (ed. Olle, T.W. and Verrijn, A.A.E.), Maastricht, 1994, North–Holland, Amsterdam, pp. 169–194.

Harmsen, F., and Brinkkemper, S. (1995) Design and Implementation of a Method Base Management System for a Situational CASE Environment, in *Proceedings of the 2nd Asia–Pacific Software Engineering Conference (APSEC'95)*, IEEE CS Press, Brisbane, December 1995, pp. 430–438.

Heym, M. and Osterle, H. (1992) 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., pp. 142–155.

Kaplan, S.M., Tolone, W.J., Carroll, A.M., Bogia, D.P., and Bignoli, C. (1992) Supporting Collaborative Software Development with ConversationBuilder, in *Proceedings of the 1992 ACM Symposium on Software Development Environments,* ACM Press, pp. 11–20.

Krant, R.E. and Streeter, L.A. (1995) Coordination in Software Development, *CACM,* **38** (3), 69–81.

Kumar, K. and Welke, R.J. (1992) *A proposal for situation–specific methodology construction*, Challenges and Strategies for Research in Systems Development. Wiley, New York.

Lonchamp, J. (1995) CPCE: A Kernel for Building Flexible Collaborative Process–Centred Environments, in *Proceedings of the 7th Conference on Software Engineering Environments,* IEEE CS Press, Netherlands, April 5–7 1995, pp. 95–105.

Mark V Systems Ltd (1995)*MethodMaker*, 16400 Ventura Boulevard, Encino, California 91436.

Mark V Systems Ltd (1995) *ProcessMaker*, 16400 Ventura Boulevard, Encino, California 91436.

Meyers, S. (1991) Difficulties in Integrating Multiview Editing Environments, *IEEE Software,* **8** (1), 49–57.

Nijssen, G.M. and Halpin, T.A. (1989) *Conceptual Schema and Relational Database Design: A Fact Oriented Approach.* Prentice–Hall, Englewood Cliffs, NJ.

Nuseibeh, B. and Finkelstein, A. (1992) 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., pp. 50–61.

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

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

Saeki, M., Iguchi, K., and Wen–yin, K. (1993) A Meta–model for representing software specification and design methods, in *Proceedings of the IFIP WG8.1 Conference on Information Systems Development* (ed. Prakash, N., Rolland, C., and Pernici, B.), Como, Italy.

Song, X. and Osterweil, L.J. (1992) 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., pp. 225–229.

Swenson, K.D. (1993) A Visual Language to Describe Collaborative Work, in *Proceedings of the 1993 IEEE Symposium on Visual Languages,* IEEE CS Press, Bergen, Norway, pp. 298–303.

Venable, J.R. (1993) CoCoA: *A Conceptual Data Modelling Approach for Complex Problem Domains*, Ph.D. dissertation, Thomas J. Watson School of Engineering and Applied Science, State University of New York at Binghampton, 1993.

Venable, J.R. and Grundy, J.C. (1995) Integrating and Supporting Entity Relationship and Object Role Models, in *Proceedings of the 14th Object–Oriented and Entity Relationship Modelling Conferece (OO–ER'95),* , Gold Coast, Australia, Dec 13–16 1995, Lecture Notes in Computer Science 1021, Springer–Verlag.

Vessey, I. and Glass, R.L. (1994) Applications–based Methodologies, *Information Systems Management*, 53–57, Fall 1994.

Wasserman, A.I. and Pircher, P.A. (1987) A Graphical, Extensible, Integrated Environment for Software Development, *SIGPLAN Notices*, vol. 22, no. 1, 131–142.

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

BIOGRAPHY

Dr John Grundy has been a Lecturer in Computer Science in the Department of Computer Science, University of Waikato since 1993. He holds the BSc(Hons), MSc and PhD degrees, all in Computer Science from the University of Auckland, New Zealand. His research interests include software engineering environments, software process technology, software engineering methodologies, visual programming, and object–oriented systems. He is currently developing the Serendipity process modelling and enactment environment, and using Serendipity to provide a process modelling and work coordination tool for large CSCW systems, such as collaborative software engineering environments.

Dr John Venable has been a Lecturer in Information Systems in the Department of Computer Science, University of Waikato, Hamilton, New Zealand since 1994. He obtained his PhD in Computer Science and Information Systems in 1994 from Binghamton University in Binghamton, New York, USA. He has lectured since 1983 at Binghamton University, Central Connecticut State
University, and Aalborg University, Denmark. Dr Venable's main interests are in information systems development, particularly in its practice, methods, and appropriate tool–based support. Currently, he is researching the incorporation of CSCW features into CASE tools to better support and improve the systems development process.