Working Paper Series
ISSN 1170-487X

Serendipity: integrated environment support for process modelling, enactment and improvement

**by John C Grundy,
John G Hosking**

Working Paper 96/17

August 1996

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

Serendipity: integrated environment support for process modelling, enactment and improvement

JOHN C. GRUNDY

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

JOHN G. HOSKING

Department of Computer Science,
University of Auckland,
Private Bag 92019, Auckland, New Zealand
email: john@cs.auckland.ac.nz

Abstract. Large cooperative work systems require work coordination, context awareness and process modelling and enactment mechanisms to be effective. The Serendipity environment provides visual languages for specifying process models and event processing. Enacted models can be modified during or after use and can act as plans of work to be done, describe work in progress, and record work done on a project. Serendipity has been integrated with an Information Systems engineering environment and office automation applications, without modification to these pre-existing tools. Animation of process models allows collaborating users to remain aware of the work contexts of their collaborators. Information about the current enacted process stage is attached to descriptions of changes made to work artefacts, recording the context of work. Such changes are also stored by the process stage, allowing collaborators to review the stage work history. Serendipity's visual event processing language allows users to specify rules and actions triggered by enactment, process or work artefact modification, or tool events. This paper describes Serendipity, our experiences using it, and its architecture and implementation.

Keywords. process modelling, process enactment, process-centred environments, work coordination, context awareness

1. Introduction

Most computerised or semi-computerised work systems have evolved informal or semi-formal process models. These attempt to describe the use of different tools on a project, the interchange and modification of work artefacts by tools and workers, and the flow of control and/or data. Workflow management systems and process-centred software engineering environments are examples of tools which have been developed to assist in the construction, use and evolution of process models. Typically these use low-level, textual process models which are difficult for end-users to understand and modify. They also lack adequate HCI support for cooperative work (Krishnamurthy, 1995), and use inflexible process models which inadequately handle exceptions and are difficult to change while in use (Swenson, 1993).

Computer-Supported Cooperative Work (CSCW) systems provide cooperative editing tools to support collaboration. These include synchronous editors and tools, such as shared workspaces, telepointers and videoconferencing, and asynchronous communication, such as email and version control systems (Ellis, 1991). These systems have a low-level focus (Krishnamurthy, 1995), and lack support for general work process modelling. Some systems, such as ConversationBuilder (Kaplan, 1992, Krishnamurthy, 1995), attempt to bridge the gap between CSCW tools and process modelling tools but with only limited success.

Existing CSCW and Process Modelling systems are not integrated well with available work tools, such as document editors, CASE tools and programming environments (Tolone, 1995). This usually means custom-built systems must be developed which exhibit CSCW behaviour, or existing tools modified to permit (often loose) integration with process modelling tools.

We describe Serendipity, which integrates process modelling and cooperative work support for large collaborative systems. It provides, graphical process modelling facilities which act as both general process models and plans for and histories of work. A graphical language specifies event handling for process model enactment and work artefact modification events. Serendipity provides a work context (the "current enacted" process stage) and work context awareness (highlighting collaborators' work contexts). Work artefact changes are annotated with a work context and stored by the current enacted process stage, along with process enactments, forming work and enactment histories. Serendipity is integrated with existing tools via an event passing system, necessitating no modifications to these tools. Serendipity has been applied to collaborative process modelling and work planning, process improvement, method engineering and office automation.

2. Problem Domain: Large Cooperative Work Systems

Recent Computer Supported Cooperative Work (CSCW) research has primarily focused on low-level interaction mechanisms, such as synchronous and asynchronous editing. Examples include most Groupware systems (Ellis, 1991), GroupKit (Roseman, 1992), Mjølner (Magnusson, 1993), C-MViews (Grundy, 1995), and Rendezvous (Hill, 1992). These systems lack information about the "work context" changes have been carried out in. Little or no support is provided for modelling work processes, planning work, or grouping changes into histories based on particular work tasks. Some work has been done on providing higher-level process modelling and coordination facilities, such as workflow configuration (Medina-Mora, 1992), software

process protocols (Kaplan, 1992), and shared workspace awareness (Roseman, 1992). These systems are generally separate from the work artefacts or editing tools and are not used to provide work context information.

Process-Centred Software Engineering Environments (PCSEEs) define the processes used to construct software systems. Examples include Merlin (Peuschel, 1992), Marvel (Barghouti, 1992), CPCE (Lonchamp, 1995), and EPOS (Jaccheri, 1992). Computer-Aided Method Engineering (CAME) tools support evolution of the methodologies and notations used for software development. Examples include Decamerone (Harmsen, 1995) and MethodBase (Saeki, 1993). Most such environments use complex, textual descriptions of processes and tool configurations which are difficult to understand and modify, often can not be modified while in use, and have poor exception handling. Most lack adequate specification of coordination between process stages and external, non-computerised processes. Some provide high-level, graphical process modelling tools but most lack arbitrary event handling (from either the process model or work artefacts). A few provide triggering mechanisms which carry out actions based on some event. Rule-based PCSEEs provide complex, textual rules which specify constraints over process models. These approaches are low-level and often difficult for users to understand and modify. This is particularly true in process modelling domains such as office automation, where unsophisticated end-users do not understand these triggering or rule-based languages. Most PCSEEs are not well integrated with existing, non-process-centred software development tools (Marlin, 1993).

Workflow systems model, usually graphically, the flow of documents between tasks and subtasks for a problem domain. Examples include Action Workflow (Medina-Mora, 1992) and Domino (Kreifelts, 1991). This approach is not well suited for work process modelling. Often exceptions to the workflows outnumber useful cases, workflow models can't be modified during use, and support for arbitrary event handling (for example, from work artefacts or tools) is limited. Regatta (Swenson, 1994) uses a visual planning language to describe work processes. This lacks descriptions of roles, work artefact and tools used in the work process, however, and does not support event-handling and process rules.

Some work, including ConversationBuilder (Kaplan, 1992), MultiviewMerlin (Marlin, 1993), and Worlds (Bogia, 1995), attempts to bridge the gap between workflow/PCSEEs and CSCW. These have only had limited success, due to continuing problems of integrating existing tools into the environments and the limitations of the process modelling tools used (Marlin, 1993, Bogia, 1995).

In summary, for large cooperative work systems, process modelling, coordination of cooperative work activities, and low-level editing and communication mechanisms are needed. The ability to collaborate in the modelling and improvement of work processes and the planning and documentation of work is also needed. Users need to be aware of the contexts (i.e. process stages) in which other users' work is carried out. Environments should encourage users to build good work process models, plans and histories. Users should be able to easily specify the process model and work artefact changes they are interested in, and how they are informed of them. The process modelling and work planning aspects of such environments must be well-integrated with the tools used to view and modify work artefacts.

3. Process Modelling and Work Planning with EVPL

Serendipity provides a high-level, visual language for describing process models. This is an adaptation and extension of Swenson's Visual Planning Language (VPL) (Swenson, 1993), used in the Regatta project (Swenson, 1994) for constructing plans and subplans for work tasks. VPL records plan enactments, supports restructuring of plans while in use, and actioning of partially defined plans for later completion. VPL provides only basic task modelling capabilities, lacking representations of roles, work artefacts, tools, or arbitrary event processing. There currently appears to be little integration of Regatta with the CASE and office automation tools used to perform work.

We have developed the Extended Visual Planning Language (EVPL), which preserves the notion of "work plans", but adds modelling capabilities to support process modelling. EVPL can thus be used to both model generic, reusable work processes, and to model and record work plans and histories for a particular project.

Figure 1 is a process model for updating a software system (in window "m1:model1-process"). This would normally be part of a larger process, such as the ISPW-6 software process example (Kellner, 1990). Several *stages* are shown, each describing part of the process. Each stage contains a unique *id* (e.g. "m1.1"), the *role(s)* which carry out the stage (e.g. "designer"), and the *name* of the stage (e.g. "design changes").The id is user-defined and uniquely identifies stages when they appear in multiple views of the process model. Enactment *event flows* between stages can be labelled or unlabelled, the label being the *finishing state* of the stage the flow is from (e.g. "finished design"). The "m1.2:implement changes" stage has a shadow, indicating multiple implementers can work on the stage (i.e. it has multiple subprocess enactments). *Start stages* (e.g. "start changes") indicate where the process model enactments may begin, and *finish stages* (e.g. "changes approved") indicate the possible finishing states of the process.

Each process stage can be expanded to describe *subprocess* model definitions. Underlined stage IDs/roles (for example, m1.1) indicate the presence of a subprocess model for a stage. For example, "m1.1:design changes" defines a subprocess model which checks out the design to be modified from a shared repository, determines modifications to the design, and then checks it back in ("m1.1:design changes-subprocess"). The two "check out design" stages are copies of a reusable *template* process model, as indicated by italicising their IDs/roles.

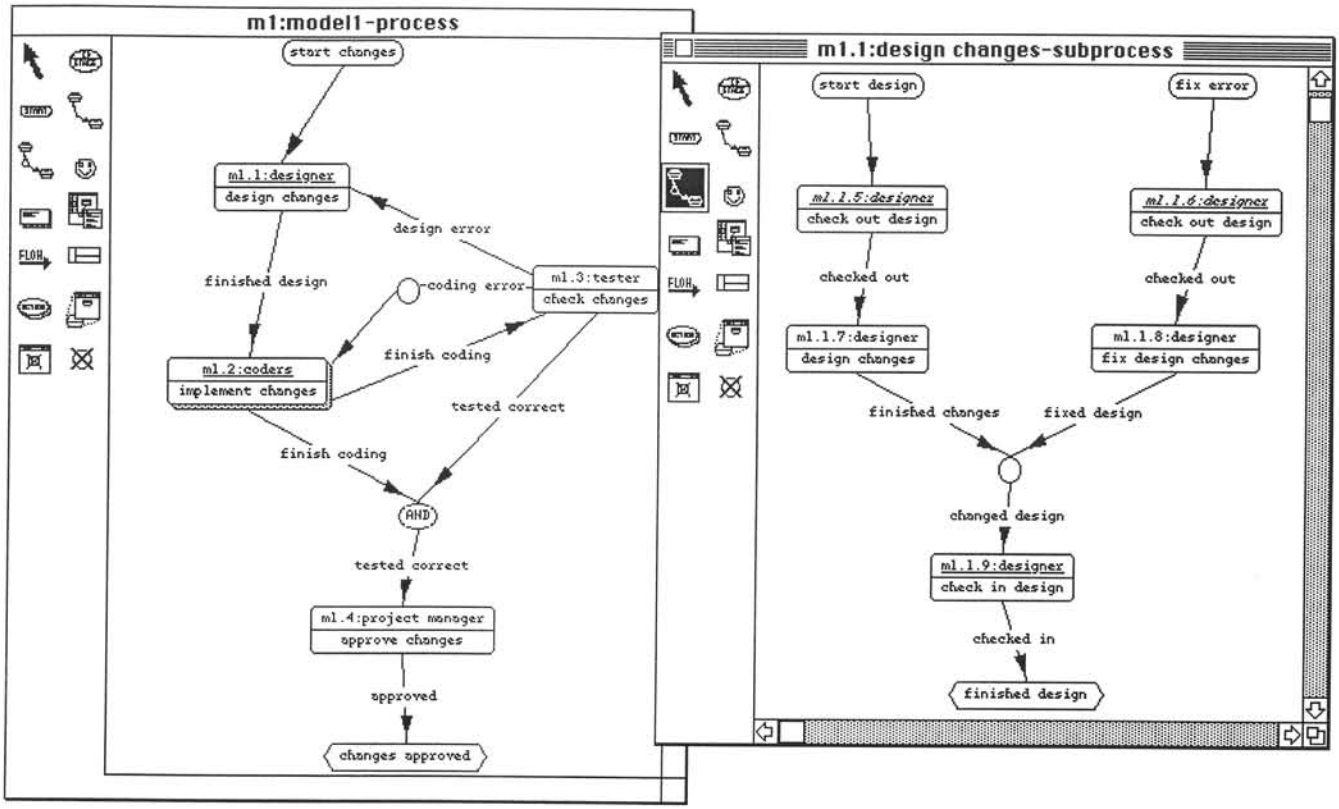

Figure 1. A simple software process model and subprocess model in Serendipity.

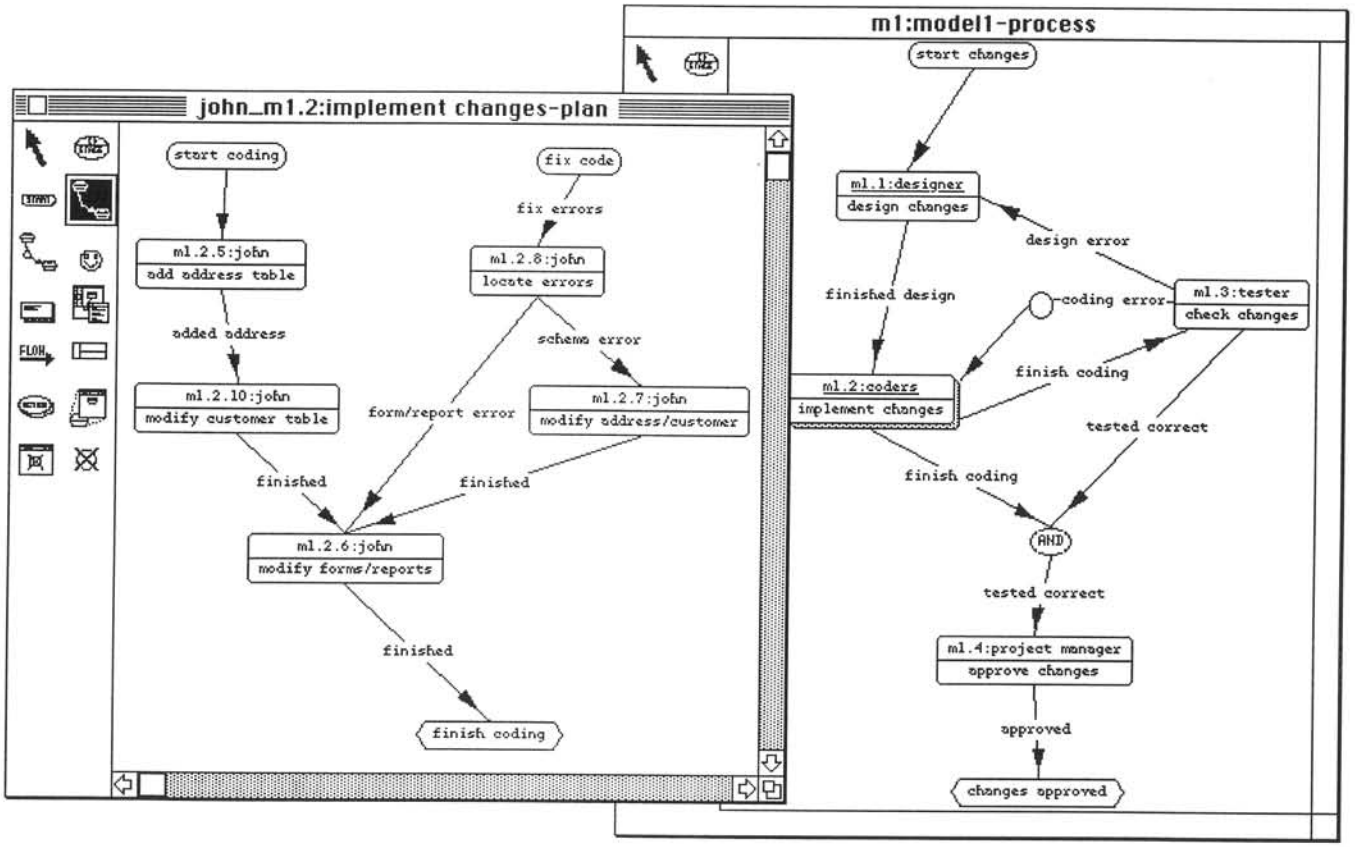

Figure 2. A work plan for coder "john".

The *AND stage* between m1.2, m1.3 and m1.4, implies stages m1.2 and m1.3 must both finish in the given finishing states before m1.4 is enacted. This does not necessarily mean all subprocesses of m1.2 or m1.3 need be finished, as they may have parallel flows which terminate with different finishing states at differing times. However, at least one flow must terminate m1.2 and m1.3 in the specified finishing states for m1.4 to be enacted. The *OR stage* (unlabelled circle) iconnecting m1.1.7, m1.1.8 and m.1.9, indicates m1.1.9 is enacted when either m1.1.7 or m1.1.8 finish. The OR stage could be omitted and the

flows from m1.1.7 and m1.1.8 be flowed directly into m1.1.9, as input flows are implicitly ORed. The OR stage does, however, act as a useful notational convenience.

To use this process model, stages in the model are *enacted*. When a stage completes in a given state, event flows with this state name (or no name) activate to start the linked stages. Enactments of each stage are recorded, as are the work artefact changes made while a stage is the *current enacted stage* for a user (i.e. the user's work context). Section 5 further describes process enactment.

Empty, or leaf, stages have no definition of their work process. In addition, some process model data, such as roles, may be abstract, requiring specification for a project. Users working on a particular project can define *work plans* for leaf stages, and extend models to more precisely specify role, artefact and tool data. For example, the process model above has work plans defined for each "coder" for stage m1.2. The people filling these "coder" roles can define their own work plans or have one defined by some other role (perhaps the "designer" or "project manager").

Figure 2 shows the EVPL work plan definition for the "m1.2:implement changes" stage for coder "john". This specifies that when this stage is enacted with "start coding" (from m1.1), "john" will add a relational database table "address", modify a table "customer", and then modify any forms and reports affected by these schema changes. If the stage is enacted with "fix code" (from m1.3), then "john" must locate the error and fix the schema, forms and reports. Different work plans can be defined for other people filling the "coder" role. This illustrates the versatility of EVPL for both defining work process models and planning the actual work to be done on a particular project.

Note that the mappings of the "finished design" and "coding error" finishing states of m1.1 and m.1.3 to the "start coding" and "fix code" starting state names are specified in a dialog when adding each enactment event flow. This allows stages to be enacted in the same start state by multiple event flows from other stages, such as for m1.2.6 in figure 2.

The process model of Figure 1 is insufficient to fully describe work processes, as it lacks descriptions of tools and artefacts used, and coordination between stage roles. Figure 3 shows a different perspective (view) of the process model. This retains the process stages but not the enactment event flows. Instead, *usage flows* describe the tools, artefacts and roles that stages use. For example, "m1.1:design changes" uses the "ER Modeller" tool and the "ER design" artefact. The usage flow between "ER design" and "ER Modeller" indicates the latter is used to modify the former. The usage connections to the "designer" role from m1.1 and m1.3 indicates coordination between the m1.3 role ("tester") and the m1.1 role ("designer"), in this case the roles must communicate ("talks with").

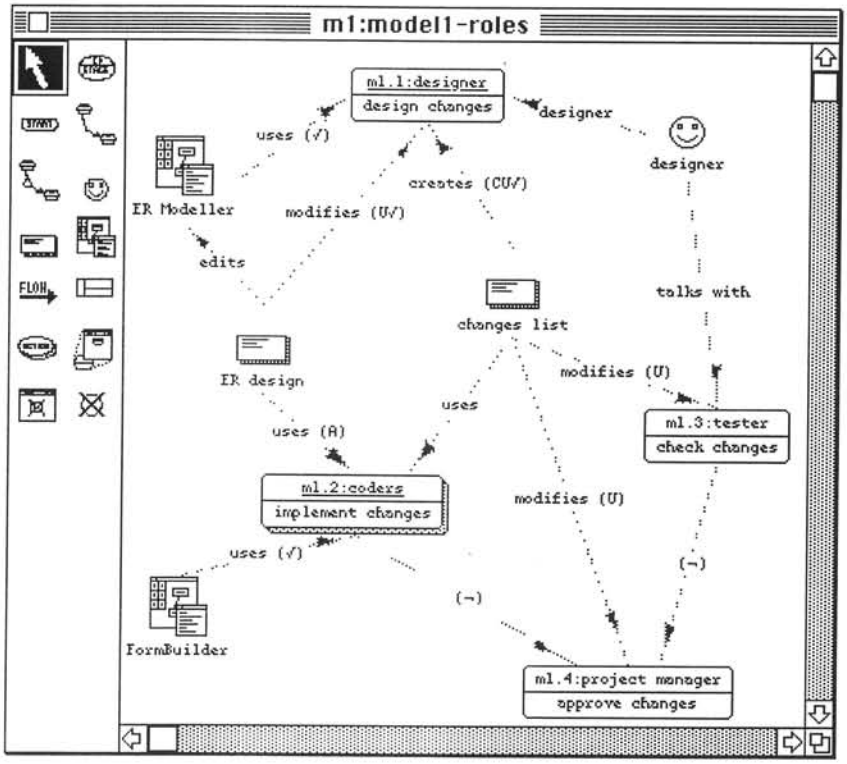

Figure 3. A data, tool and role-oriented perspective of the first process model.

Additional modelling capabilities provided by EVPL include a subtyping connection for building artefact, tool or role hierarchies. Usage connections can be annotated to describe: whether data is created (C), accessed (A), updated (U), or deleted (D); whether only the tool(s), artefacts or roles defined (√) can be used; and whether a stage cannot use a particular tool, artefact or role (¬). If a usage flow links two stages, "√" specifies the stages may be enacted at the same time, while "¬" specifies the

stages can not be enacted at the same time. Process modellers can choose to add such annotations to more explicitly define and enforce process models, or omit them and use the model to guide the work process.

4. Defining Event Filters and Actions with VEPL

The process models and work plans described in the previous section are fairly static. Stage enactment flows are described, indicating the flow of enactment events between stages, but no other event handling is specified. In contrast to the complex, textual approaches of PCSEEs, Serendipity's Visual Event Processing Language (VEPL) visually specifies event handling and process model rules. VEPL provides *filters*, which receive events from stages, artefacts, tools or roles, and determine if the events match user-specified criteria. *Actions* receive events, usually from filters, and carry out some user-specified action in response to the events. The language is fully integrated with EVPL concepts and thus provides a graphical, high-level specification of event handling for EVPL process models and work plans.

Figure 4 shows a simple extension to the process model in Figure 1. In this model, the testing (stage m1.3) can be carried out while further design and/or coding takes place. To coordinate the people associated with the "designer", "coders" and "tester" roles, Figure 4 specifies when notifications are sent to coders that testing has been completed or is in progress.

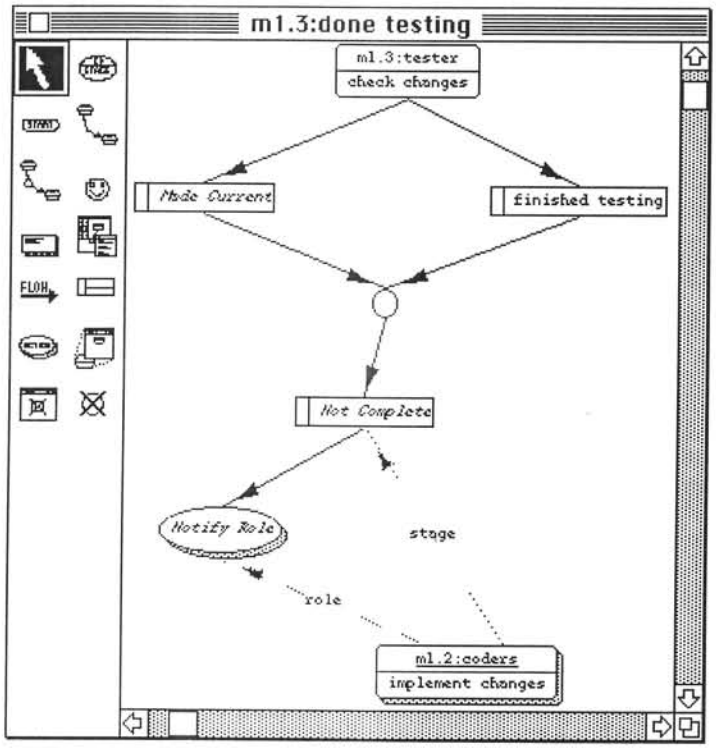

Figure 4. An example of process stage event filtering.

The lefthand event flow from m1.3 specifies all enactment events on m1.3 should flow into the filter "Made Current". This checks if the received event shows the attached stage, m1.3, has been made the user's current enacted stage (i.e. the user is now working on this stage) and, if so, the event flows into the OR stage and on to "Not Completed"; otherwise event propagation stops. "Not Completed" ensures the stage it is linked to by usage connection "stage", m1.2, is either enacted or able to be restarted (not yet completed). If so, the event flows into action "Notify Role", which informs the person or people filling the role of the stage linked to it by usage connection "role" (in this example, the "coders"), that testing has started or finished. The person or people filling the role would usually be informed by an e-mail like message, but users can specify they are informed by the opening of a dialog on the coders' screen, shading the "m1.2:implement changes" stage icon to indicate presence of a message, etc (Grundy, 1996).

The righthand event flow from m1.3 notifies coders if testing has finished in either the "fix code", "fix design" or "finished testing" states. The coders are again only notified if the m1.2 stage has not completed. Note that this event flow could also be described by omitting the filter and having three named event flows ("fix code", "fix design" and "finished testing") flowing into the OR stage i.e. named event flows themselves are simple filters which propagate finishing events matching their name.

This example illustrates how filters and actions can be parameterised by named usage flows. They can also contain user-specified data. For example "Notify Role" has a flag specifying whether email, icon shading or a dialog is used. Filters and actions can be reusable ("Notify Role", "Made Current" and "Not Complete") or specific to a particular process model ("finished testing"). Specific filters can be defined by a pattern-matching language (if simple filtering), supplying data for a reusable template (via a forms interface), or use of an API to the implementation language of the Serendipity environment.

Event filtering and actioning is useful for processing arbitrary events from artefacts, tools and roles (users). For example, figure 5 specifies that the "designer" associated with "m1.1:design changes" is to be notified of any schema updates made by "coders" associated with "m1.2:implement changes". Artefact updates from any user filling the "coder" role during m1.2 flow into a filter ("RDB table change"), which checks if the change was to a table. If so, the "designer" is notified. The event flow annotation "Æ" indicates artefact events and not stage enactment events are of interest. The "Σ" symbol indicates hierarchical interest i.e. this filter applies not only to m1.2 but also all of m1.2's subprocess stages. "RDB table change" can be defined in two ways: a large list of artefact change event patterns which indicate the change is to a table, or by determining if the event originated from the RDB table designer tool.

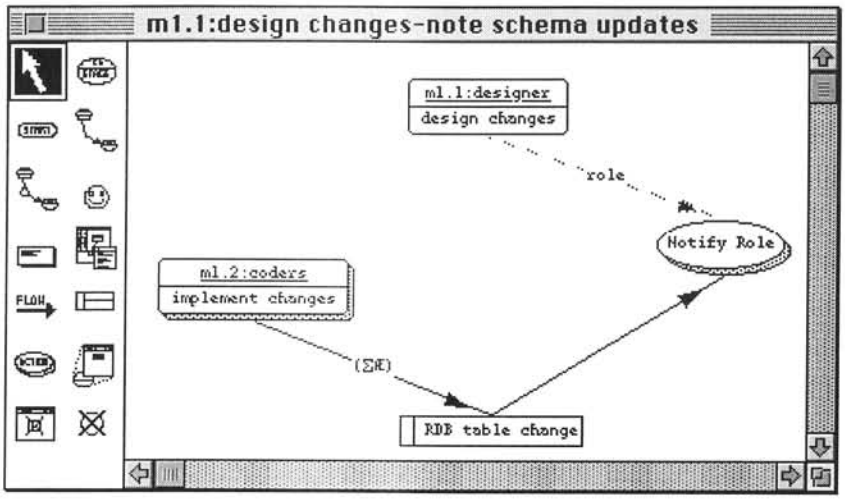

Figure 5. An example of artefact change event filtering.

Roles, tools and artefacts can act as filters. Figure 6 specifies that "rick" is interested in any changes "john" makes to the "customer" table, and "rick" is to be informed asynchronously of these changes by storing them in a change list. Changes from m1.2 flow to the "customer table" artefact, which here acts as a filter, only passing on change events for this artefact. Filter "Made By" checks the changes were made by user "john", and action "Store Event" records the event in a change list, named "john's changes" for user "rick". Artefact update events can also be filtered through roles and tools. The m1.2 stage can be removed from the example in figure 6, and the flow from artefact "customer table" annotated with Æ. This would then specify that any change at all to "customer table", made by "john", should be stored for "rick", regardless of the plan stage it was made in.

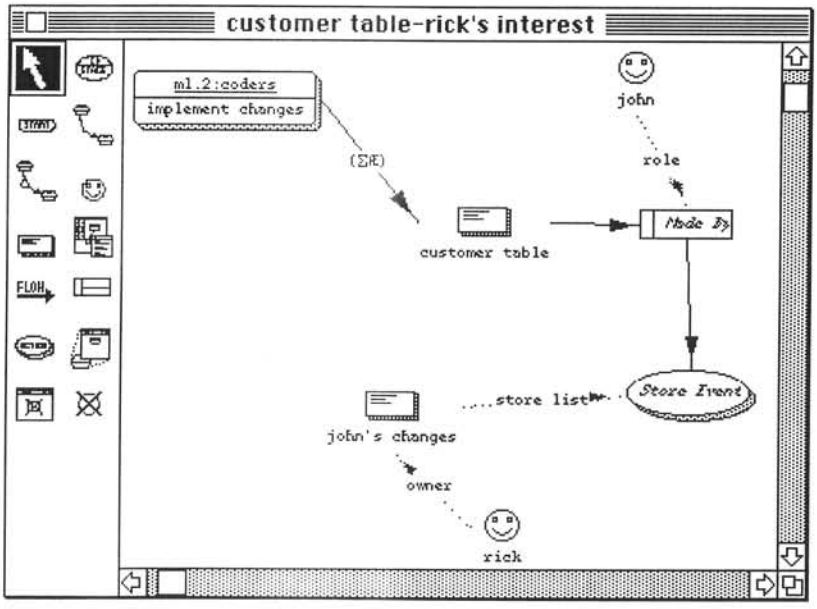

Figure 6. An example of complex filtering and actions.

VEPL filter/actions can be expanded into subprocesses, parameterised by start and finish stages. Figure 7 is a generic filter/action which checks if a change to a given entity (input "entity") is a rename or attribute modification (add, delete, rename or change type), and, if so, stores the artefact change event in an "updates list" specified for a given "stage". The finish stage "update" represents an output from this filter/action, allowing the event to be flowed onto other filters/actions. The filter/action can be reused by specifying "entity", "stage" and "updates list" values via usage connections to an entity artefact, stage and updates list artefact respectively. The output "update" can be used if required or left unused.

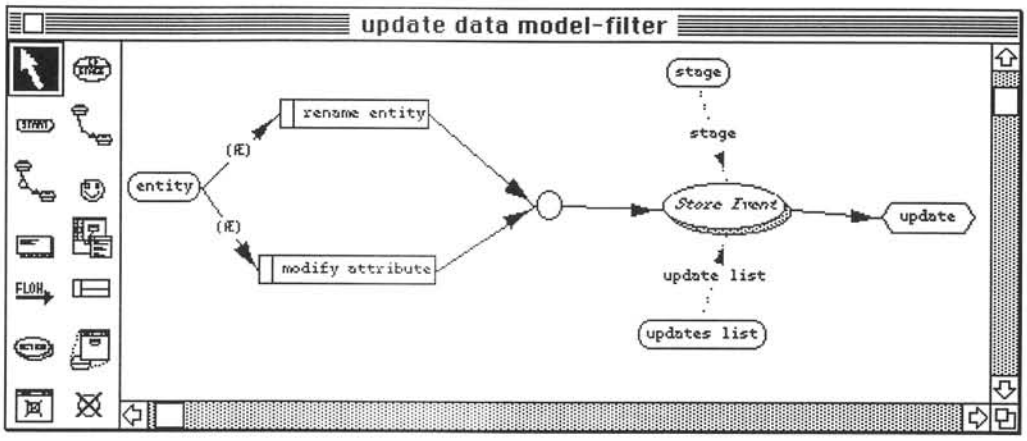

Figure 7. A parameterised filter/action.

Event flows can be annotated to indicate multiple events from a stage, artefact, tool or role are to be processed, rather than the single event processing example given above. A "*" indicates the filter/action can receive multiple events before deactivation, allowing it to recognise and process complex event sequences. Such filter/actions store state data for multiple events flowing into them, and thus can act as semi-autonomous agents. These filter/actions reset their state after receiving the sequence of events they are interested in by executing a "Reset State" action. For example, in Figure 8 the "Add Entity & Rel/Roles" filter recognises a sequence of ER updates (addition of a new entity, relationship, roles and attributes). Coders are notified of such a major schema update using a single abstract message, rather than a long list of discrete entity/relationship/role/attribute addition change messages. The "Add Entity & Rel/Roles" filter continues to receive ER modeller artefact update events until it notices an unrelated artefact is being modified, then passes on a single event to "Notify Schema Affected" and resets its state.

Other example uses of complex filters/actions include the "SynchronousEdits" action, which in figure 8 is used to specify how work coordination between people filling the roles for two concurrently enacted stages is to be achieved. It determines that the if m1.1 and m1.2 stages are concurrently enacted and both use the "RDB Table Designer" tool, the stage roles must use the synchronous editing mode of the "RDB Table Designer" tool to ensure schema changes they are making do not conflict. The "Seek Client Approval" action models a non-computerised process, and asks the "project manager" to liaise with the customer and gain approval for the new system changes made.

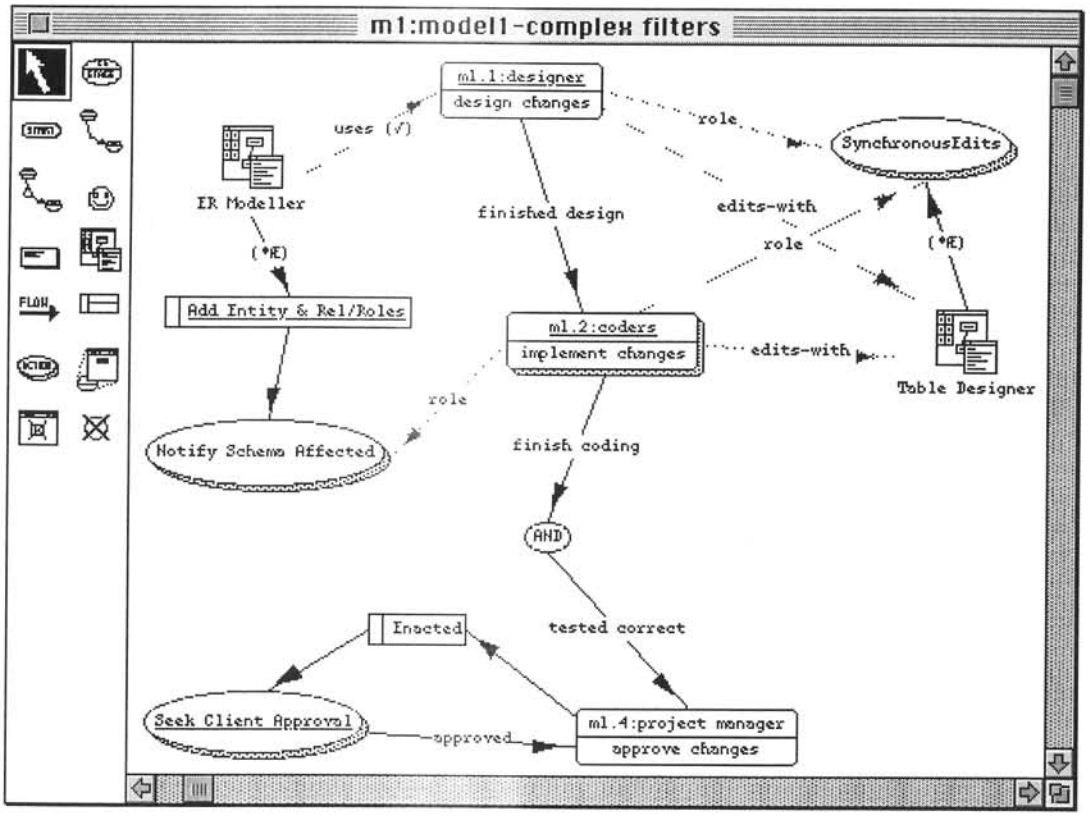

Figure 8. Multiple event processing, stage coordination and external process interfacing.

We are currently developing a third visual notation for Serendipity, the Visual Query Language (VQL). This will allow users to specify complex, graphical queries over the environment repository. It will also allow such queries to be used to define

complex artefacts for use by VEPL filter/actions. Triggering of filters and actions will then be possible over the query result, which will be incrementally updated as data is modified.

5. The Serendipity Environment

We have developed an environment for Serendipity supporting EVPL and VEPL. This provides multiple views of process models, allows processes to be enacted, supports process improvement and reuse, and allows meta-process models to be defined to describe and control the process modelling process itself.

5.1. MULTIPLE PROCESS MODEL VIEWS

EVPL and VEPL process model descriptions are produced using the same editing tool. As seen previously, several views of the same process model can be defined, each adding more information about the model as a whole. For example, figure 1 is a process model for software system enhancement, figure 3 describes the artefacts, tools and roles used this process model, and the figures in the previous section associate filters/actions with these stages, artefacts, tools and roles. Textual views of process stages are also available, which can contain comments and a description of the stage (Figure 9).

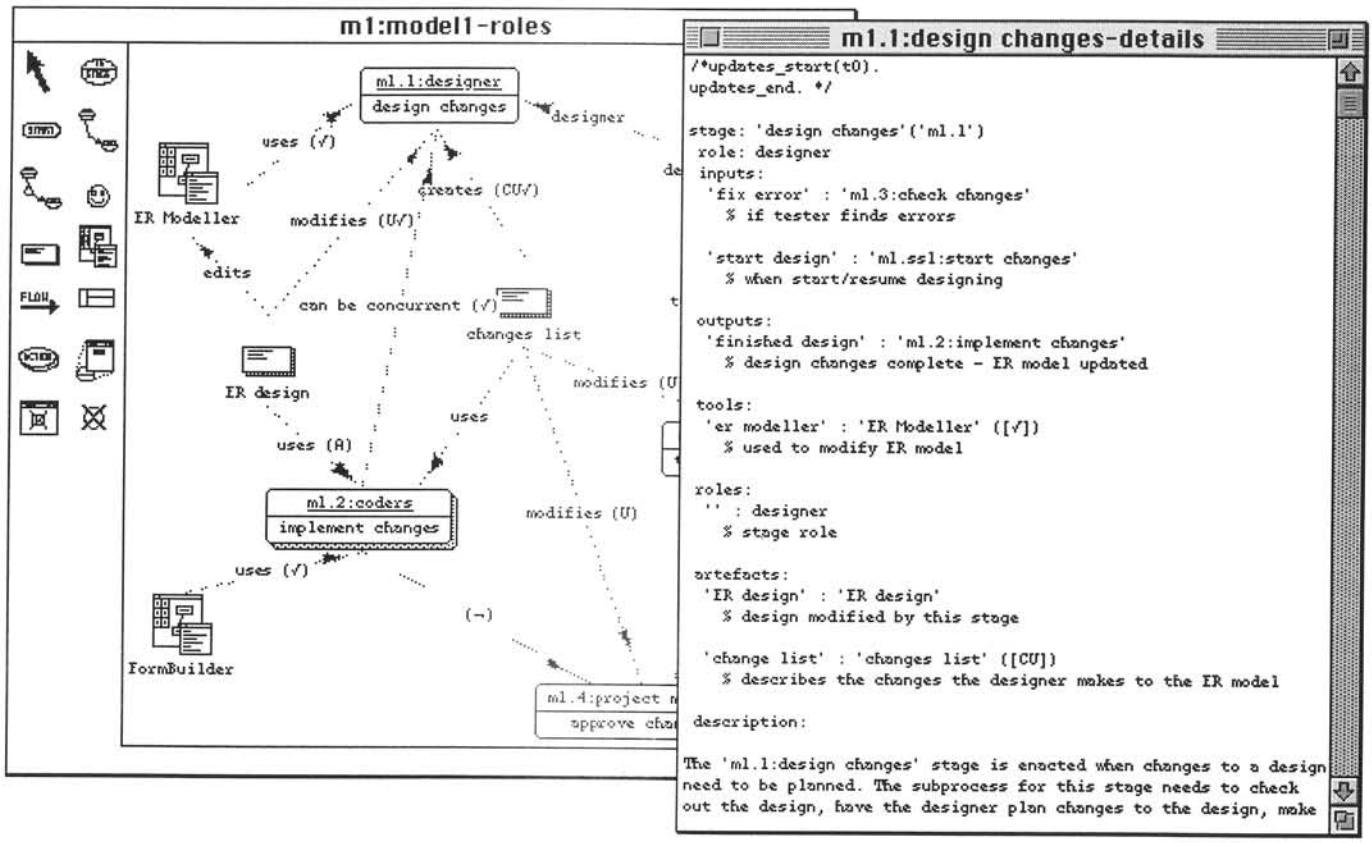

Figure 9. An example of Serendipity textual stage views.

Serendipity keeps all views of a process model consistent under change. For example, renaming stages, artefacts etc. in one view results in shared data being updated in other views. It is not possible for the environment to automatically implement some changes, as the result of the change may be ambiguous or indeterminate. For example, deleting a subprocess start stage means an event flow into the process with the same name must be either deleted or renamed to another existing start stage. In such situations, Serendipity shades the affected icons/connections to indicate the inconsistency, users browse a change description for the inconsistency in a dialog, and then resolve it.

5.2. PROCESS ENACTMENT

A Serendipity process model can be enacted for a project at any time with roles assigned to particular users or groups of users. Enacted processes can be modified at any time to extend and improve the process specification, more precisely define the work to be done, or rewrite the history of work done to assist in process improvement and work documentation.

Figure 10 shows an enacted process model. Enacted stages (those which have been started and not yet finished) are shaded. The user's current enacted stage (the stage the user is currently doing work on) is bold highlighted. Users can select any enacted stage to be their current enacted stage. Stages can be started and finished by the user (if no actions prevent this), by event flows from other stages, or by actions. Event flow(s) which have activated a stage are highlighted.

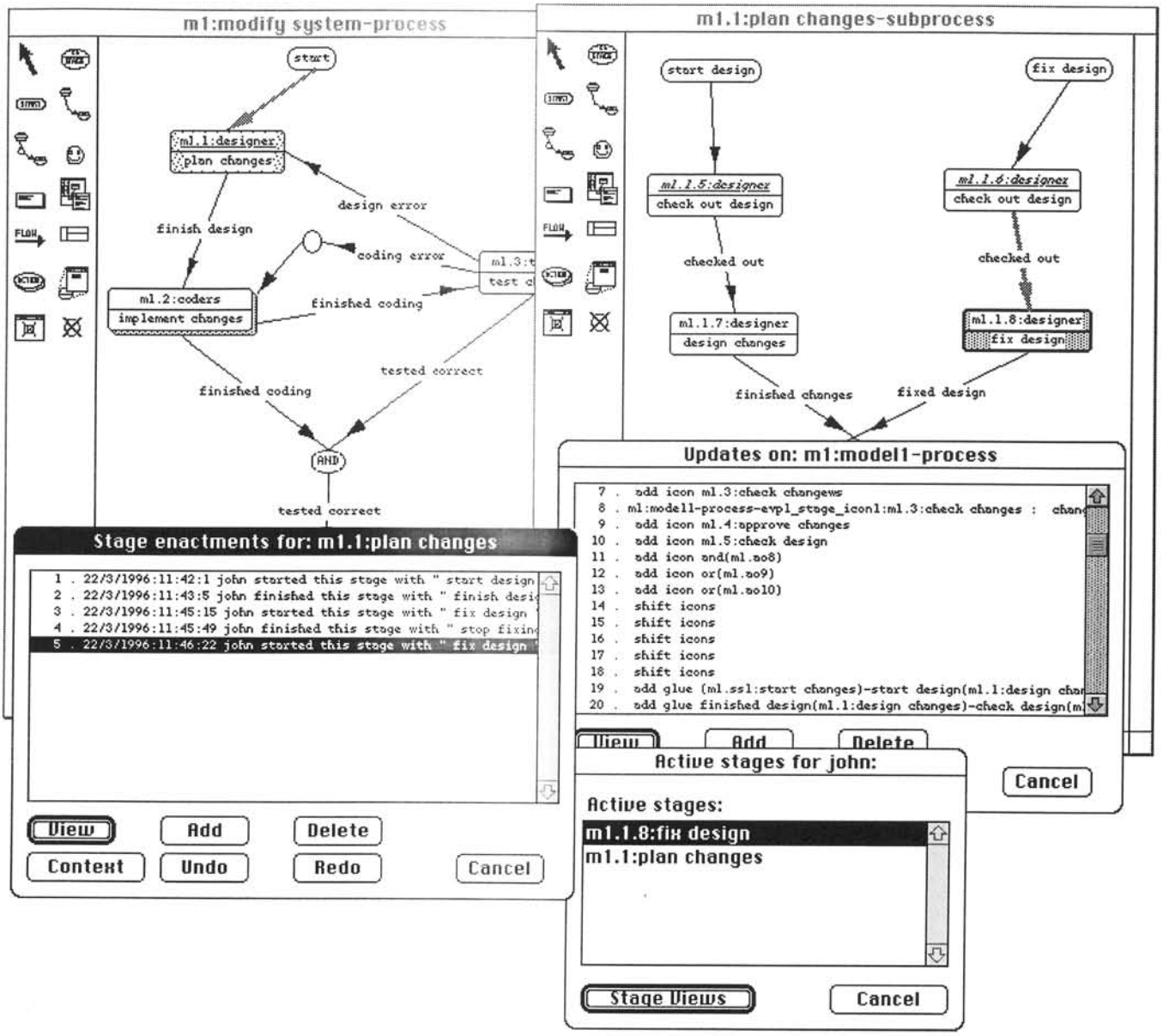

Figure 10. Enactment and modification histories for a stage, and a user's to-do list.

As work on a stage is completed or a subprocess completes (event flow reaches a finish stage), finished events flow into connected stage(s), enacting them. AND stages wait for all stages which flow into them to complete; OR stages enact all stages they flow into upon receipt of a finished event from any input stages.

When a stage is enacted, finished or set/unset as the current enacted stage, it records this event in an enactment history. A reason is stored as to when and why the event occurred and which user or stage caused the event. A modification history describing changes to each stage and view is also stored, allowing users to review the evolution of process models. Figure 10 illustrates these two histories for stage m1.1. Serendipity allows users to view all of their enacted stages, presenting these in a dialogue, which functions as a "to-do" list, also shown in figure 10.

5.3. PROCESS IMPROVEMENT AND EXCEPTION HANDLING

A process model may need modification during or after use. Models evolve over time as users become more proficient at describing their work processes. Sometimes models can't be exhaustively described, as the details of the process used varies from project to project. Process modellers may leave stages unexpanded or define general role, artefact and tool data for instantiation. Exceptions to the model may occur during use, and the model must be modified to take these into account or the exceptions handled and documented.

Serendipity allows process models to be modified at any time: before, during or after use. Modification before use allows project-specific work tailoring before enactment. Modification after use allows users to review a model's performance and improve it for subsequent reuse, and to "rewrite history" i.e. restructure the model, including enactment histories, to better document the work done.

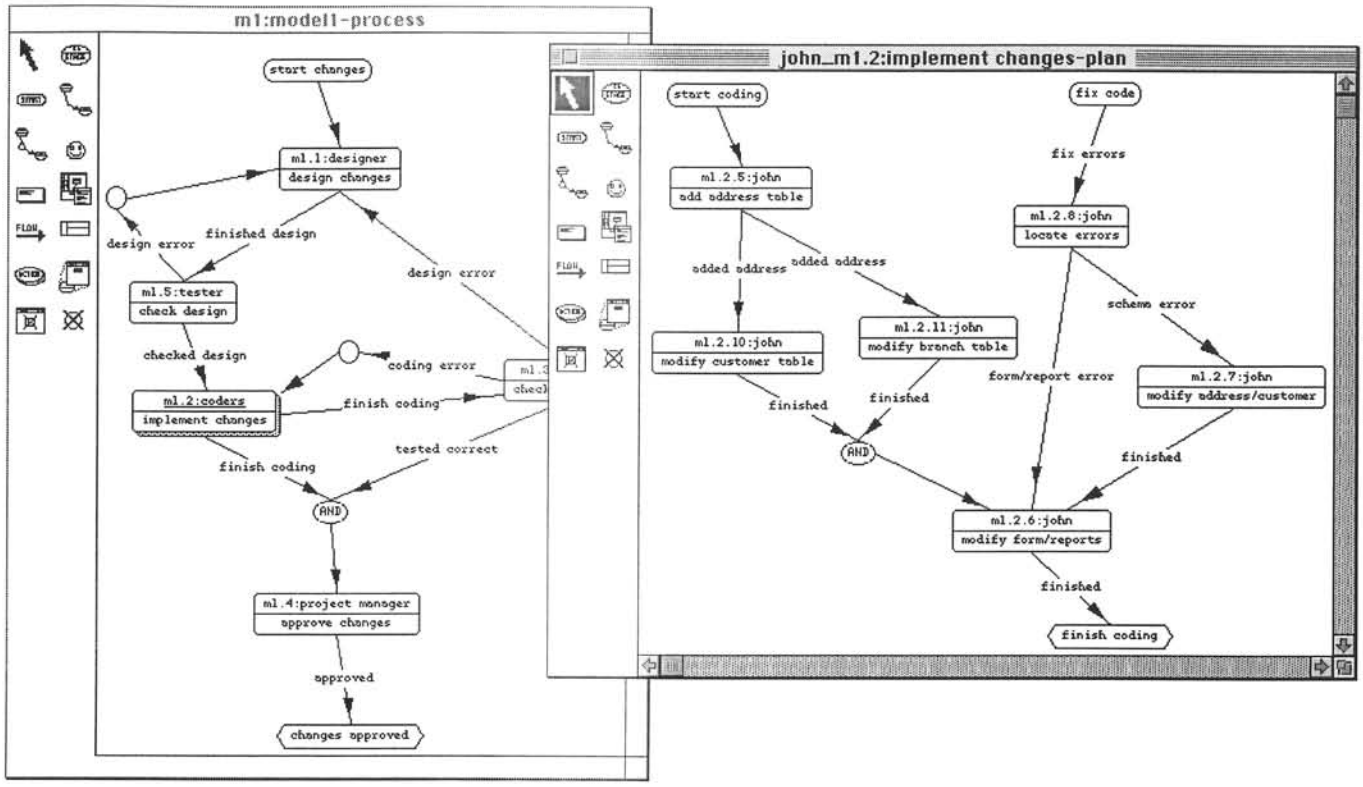

Figure 11. Simple process model and work plan improvements.

Modification during use supports refinement of project-specific work plans and data and exception handling. It also supports incremental process improvement i.e. small, bottom-up improvements to process models. Users often extend process models and work plans to define project-specific filters/actions for stages. These specify such behaviour as requesting notification of particular artefact changes, changes other users make, or enactment changes to stages other users are working on.

Figure 11 shows an example of process model and work plan improvements. 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.

Serendipity provides metrics facilities to aid users in improving process models. Users may request statistics about the complexity (number of substages, number of event/usage flows, etc.) of subprocess models, the number of enactments a stage or subprocess model has had, the amount of time each stage has been the current enacted stage, the number of modifications made to models, and the number of work artefact changes made while a stage or its subprocesses were the current enacted stage. In addition, users can add estimates of the amount of time and number of changes still to be done for process stages, which can be summarised for project managers. Statistics are displayed in histograms to indicate where large amounts of time are being spent, the progress of a project, and to compare different approaches to modelling a work process.

A special "EXCEPTION" action, which has no input event flows, is automatically invoked (if defined for a process model) when a stage, action or filter finishes in a state not handled by the process model i.e. no output event flows match the finishing state name (usually an error state). Rather than always complicate models with many error-handling actions and event flows, the "EXCEPTION" action is sent any erroneous finishing event by Serendipity. Multiple "EXCEPTION" actions can be defined for a process model with filters used to determine which one handles a particular exception. If left undefined for a model, a stage, action of filter completing in an unhandled finishing state will cause Serendipity to inform the user of a process model exception (via a change description in a dialog). The user can then handle the exception manually, or extend the model to take the exception into account.

5.4. REUSABLE PROCESS MODELS

We have described how Serendipity process models are defined, enacted and improved for single projects. We now describe how models can be reused to guide or enforce how work is done on multiple projects.

Any process stage or filter/action can be abstracted into a *process template*. When created, process templates are copied from a user-specified process stage, including any subprocesses, filters and actions, and all subprocess views. The copied components are given unique IDs to distinguish them from the originals, and an "instance" link created from template stages to the originals. Users can view and edit the template model, with any changes made recorded in template stage modification histories. Templates cannot, however, be enacted; templates are abstract and must first be instantiated (copied) into a new process model.

A template is instantiated by adding a stage to a process model, specifying it is based on a template. New process model stage, substages and views are copied from the template, given unique IDs, and linked back to the template stages. The user may further specify role, artefact and tool names, specialising the copied model from its template. Figure 12 illustrates this use of a template. Template "ps.cod:Check out design" has been instantiated in the m1.1 subprocess model. The user has modified the copied model to specify that the "designer" fills the template "arole" role. In this example, direct update of the copied subprocess model is not strictly necessary, as when this model is used, the "arole" role can be bound to the "designer" role via a dialog. Note that this subprocess is made up of actions, not stages. The "Check out design" subprocess is thus automatic with its actions run by Serendipity, rather than by manual enactment .

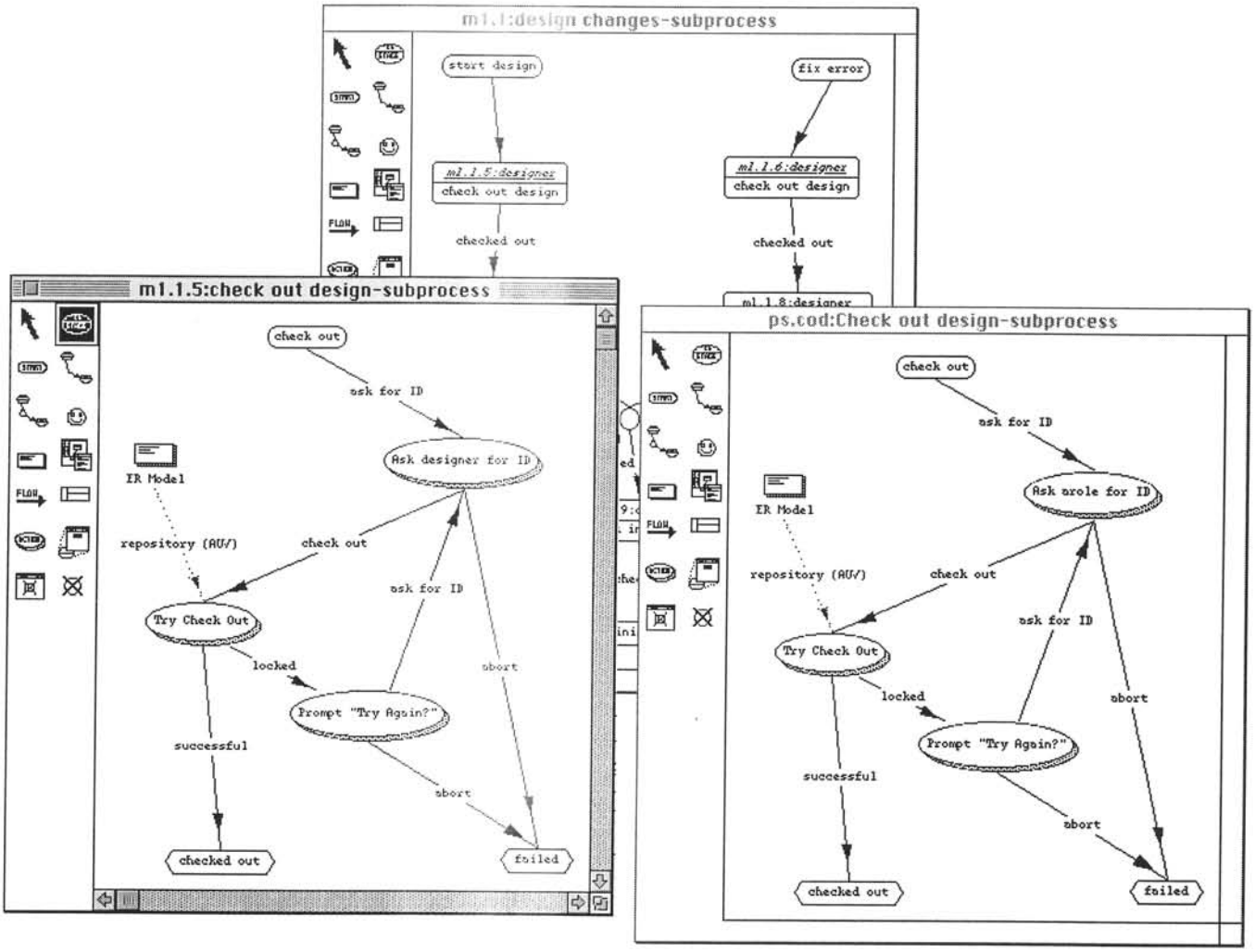

Figure 12. Reuse of a process model template.

If a template is modified, users may want the change reflected in models instantiated from the template and vice versa. Rather than manual copying, Serendipity uses the modification histories for templates and instantiations as input to the version merging system of C-MViews (used to implement Serendipity) (Grundy, 1995). This applies each change made to the template to the specified copied models where possible. Changes the version merger can not carry out, because of modification to the instantiated models after copying, are presented in a dialog. The user can then manually update the models or ignore the template change. The same process works in reverse to update a template from a modified copy.

5.5. META-PROCESS MODELS

To guide the process modelling, enactment and improvement process itself, meta-process models can be defined. These are themselves EVPL and VEPL models, but the "work artefacts" for these models are other EVPL/VEPL models, the tool is Serendipity itself, and the roles are usually project managers. Such meta-processes are useful for defining and evolving the process modelling and improvement process itself, and for coordinating this activity between multiple collaborators. An added advantage is that changes made to process models can be recorded against the current enacted stage for the meta-process model, allowing collaborators to track reasons for process model changes. Meta-process models also allow users to specify event handling and rules for the EVPL and VEPL notations themslves, by specifying filters and actions on artefact events for process models (i.e. handling events from process model artefacts and tools). Examples of such event handling/rules include specifying ownership and access privileges for process models, controlling when and how process models can be updated, and specifying default exception handling approaches.

6. Using Serendipity with Other Tools

Serendipity not only provides a flexible process modelling, enactment, improvement and work planning environment, but used with other tools it specifies the *context* of work for multiple collaborators. We have used Serendipity to support collaborative software development, method engineering and office automation.

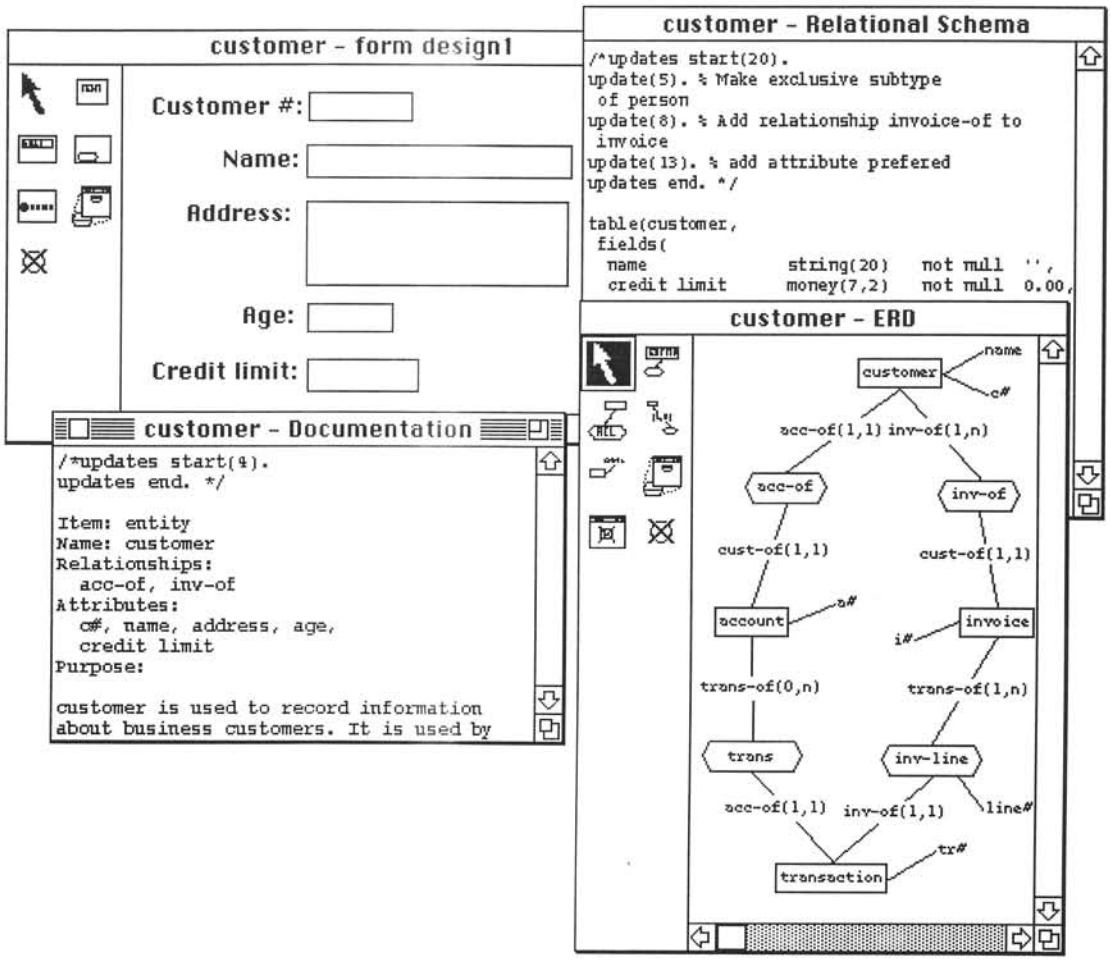

Figure 13. An integrated Information Systems engineering environment.

6.1. DEFINING WORK CONTEXTS

The current enacted stage in Serendipity defines the work context for a user. Any work artefact changes made in other tools are done to carry out the work associated with this current enacted stage. Changing the current enacted stage changes the context of work. We have used Serendipity to provide a process modelling tool for an integrated Information Systems Engineering Environment (ISEE) (Grundy, 1996). This environment includes the MViewsER ER modelling tool (Grundy, 1995), the MViewsDP form/report building tool (Grundy, 1995), and the MViewsNIAM NIAM modelling tool (Venable, 1995). A screen dump from this environment showing a simple invoicing information system under development is shown in Figure 13.

It is insufficient just to have Serendipity defining the work context for a user. When work artefact changes are made, they need to be stored with the current enacted stage to document the history of work for this stage. As shown in Section 5, stages record their enactment and modification histories. They also record the work artefact updates made while the stage was the current enacted stage. Figure 14 shows an example of changes made to a MViewsER view stored for the "m1.2.10:modify customer table" stage for coder "john".

In addition to recording work artefact updates in Serendipity, change descriptions for work artefacts stored by the tools themselves are augmented with information about the work context they were made in. This allows users of these tools to review the modification histories of work artefacts and determine what changes were made, who made them and when, and why they were made (i.e. the work context they were made in). Figure 14 shows an example of a modification history for the "customer" entity/schema from the MViewsER modelling tool. The augmented change descriptions for "customer" are also shown in its schema view, "customer - schema". Note that the changes shown in the "Updates on: customer" dialog and "customer - schema" view are customer artefact updates. Those shown in the "Artefact updates for: m1.2.10:modify customer table" are changes made to the "customer - ERD" view. The user can access the artefact-level updates made in response to these view changes via the "View" button.

Users can undo or redo work artefact changes within Serendipity or the tools which generated them. Users can also select a work artefact change and request a list of all views of the artefact in the available tool(s). In a tool, users can request a list of views of the appropriate work context of a selected change description.

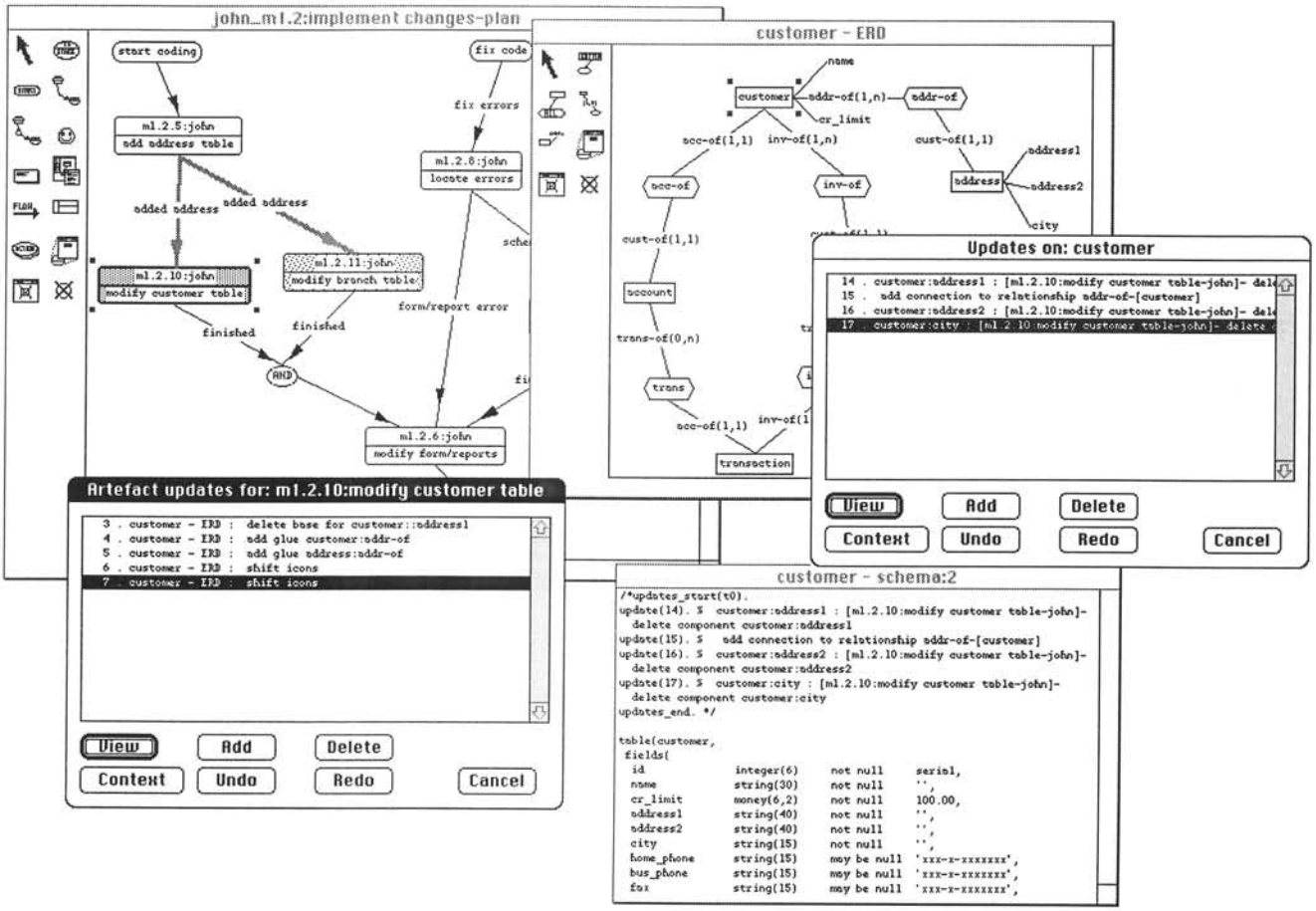

Figure 14. A work artefact change history in Serendipity and work artefact modification history in MViewsER.

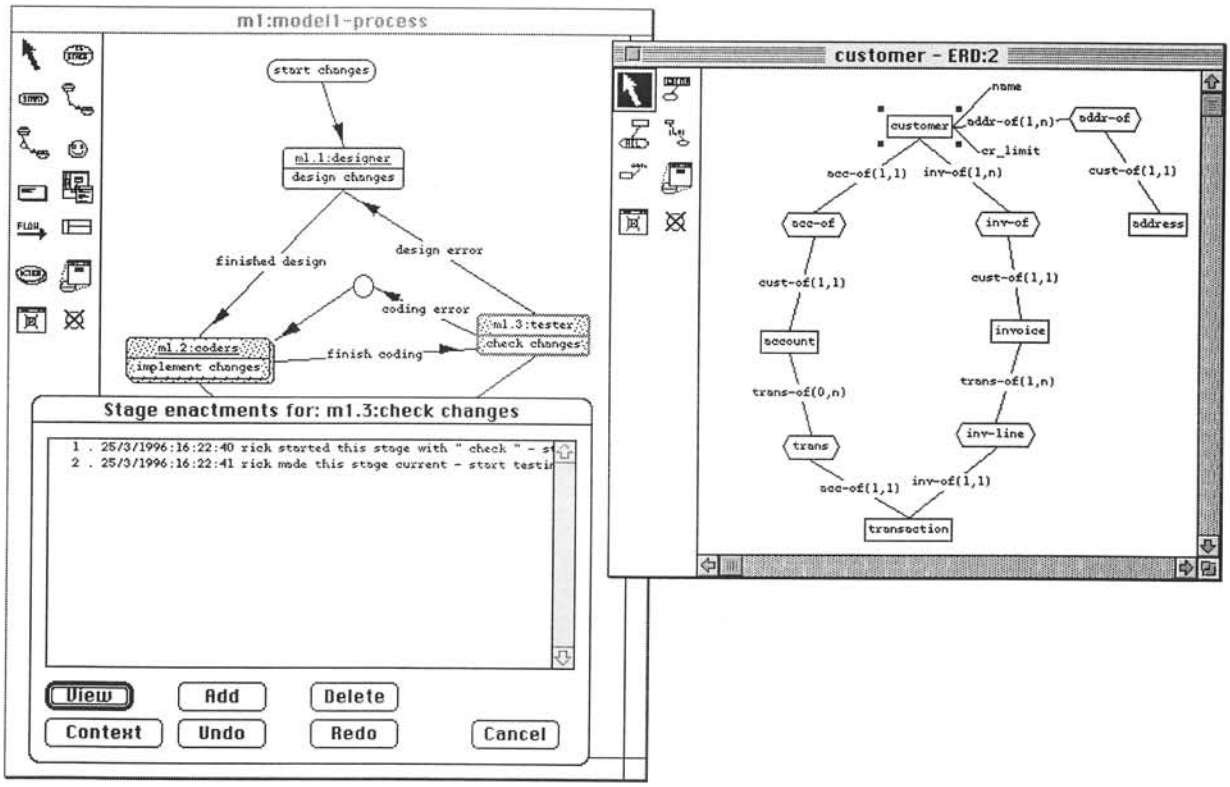

Figure 15. Supporting high-level work context awareness within EVPL views.

6.2. CONTEXT AWARENESS

Users require information about the work contexts' of their collaborators in a high-level, synchronous way. Serendipity highlights the current enacted stages of collaborators, as shown in Figure 15. The enacted stages for all users are shown, the user's current enacted stages highlighted by a bold outline, and collaborators' current enacted stages by a shaded outline. Users can specify colours with which to highlight stages, a different colour for each collaborator. Users may have more than one EVPL view open, with enacted and current enacted stages highlighted. Serendipity views may be open at the same time as work tool views so context awareness is provided while using these tools.

In addition to defining a work context for tools, used to augment work artefact change descriptions, Serendipity stages can save the "state" of a tool (windows open, artefacts in use, etc.). When a user makes a stage the current enacted stage, they can request Serendipity to restore the work context state for the tool(s) used by this stage.

6.3. COLLABORATIVE PROCESS MODELLING AND WORK PLANNING

Serendipity process model views may be shared amongst developers and synchronously, semi-synchronously or asynchronously edited. Synchronous editing uses the shared workspace metaphor where all users share the same data and all views of this data are synchronously edited (using a "what you see is what I see" editing and viewing approach (Ellis, 1991)). This allows users to closely collaborate on modelling processes and defining work plans. We have found this approach useful for defining high-level process models and for reviewing models collaboratively. It is less suitable for detailed process modelling or providing personal process models and work plans.

Serendipity also supports asynchronous process model editing, where each user has an alternative version of a shared process model view, which they modify independently of others' alternatives. Users then export their alternative to the shared workspace, and one user merges two or more alternatives to produce a new version. New merged process models can then be enacted (or reused if templates). Our process model merging approach detects inconsistencies between the merged models and presents these to the merging user for resolution (Grundy, 1995). We have found asynchronous modelling useful for users' personal process models and work plans.

Semi-synchronous process modelling is also provided. Here, users have alternate versions of a model, as for asynchronous modelling, but changes made by a user are broadcast to other interested users to be presented in dialogs or views. The other users may choose to incrementally merge the changes into their own alternative models. Asynchronous, semi-synchronous and synchronous modelling are compatible, with users able to switch between modes as required (Grundy, 1995). Semi-synchronous editing is most useful when users wish to remain aware of process model evolution being carried out by other users, but may not wish to immediately incorporate these changes into their own models.

Serendipity provides work context-dependent notes, messaging and talk-style dialogue facilities for communication, based on the MVNotes system (Apperley, 1996). Notes are asynchronous. Users add notes to work artefacts and process model stages. These are shared by all collaborators and may be perused at any time. Notes store the work context they were created and modified in, and changes to notes are recorded in stage artefact modification histories. Messages are semi-synchonous, being sent from one user to others. They record the work context they were created in and (optionally) the artefact(s) or stage(s) they are associated with. Talk-style dialogues use the work contexts of communicators to provide a context for synchronous communication.

6.4. METHOD ENGINEERING

Most software development notations and methodologies are designed to be generic across all problem domains. Research indicates, however, that configuring notations and methodologies to the needs of the particular project results in more appropriate tools and techniques (Harmsen, 1995). Computer-aided Method Engineering (CAME) tools support this by allowing system developers to specify which notations (or parts of notations) and methodologies (or parts of methodologies) are to be used for a particular project (Harmsen, 1995). We have used Serendipity to support Method Engineering for our ISEE. Serendipity EVPL views specify tool usage and process models ("methodology steps"). VEPL views specify rules and event handling which restrict the use of certain tools and guide or enforce the methodology processes. Using Serendipity in this way provides many of the facilities of CAME tools, but in a high-level, accessible notation. We are currently building a MetaCASE tool to allow developers to build, configure and improve the tools used on a project. Used with Serendipity, this will provide a full CAME environment (Grundy, 1996).

6.5. OFFICE AUTOMATION

We have used Serendipity to support process modelling for a suite of office automation programs. These include Macintosh versions of Microsoft Word, Microsoft Excel, the GlobalFax fax/OCR application, and the Eudora email utility. Two simple Serendipity process models were defined to describe the processes of ordering and receiving stock for an organisation. Ordering stock involves calculating the amount to order (using Excel), creating an order document (using Word), and sending this to a supplier (using GlobalFax). Receiving an order involves recovering data from a scanned fax or invoice (using GlobalFax's OCR software), updating stock levels (using Excel), and reporting the new stock arrival to people (using Eudora). Actions are used to launch the programs and to create, open, and save files via Apple Events from Serendipity to the running programs.

Also factored in are the non-computerised processes of checking the received stock matches the invoice list and moving the new stock to a warehouse. Figure 16 shows the "m7:order stock-process" process model and "m7.2:create order-subprocess" subprocess model views from Serendipity. The subprocess view is mostly automated. The "EXCEPTION" action succinctly handles errors generated by the "LaunchApp", "OpenDoc", etc. actions defined for the subprocess model.

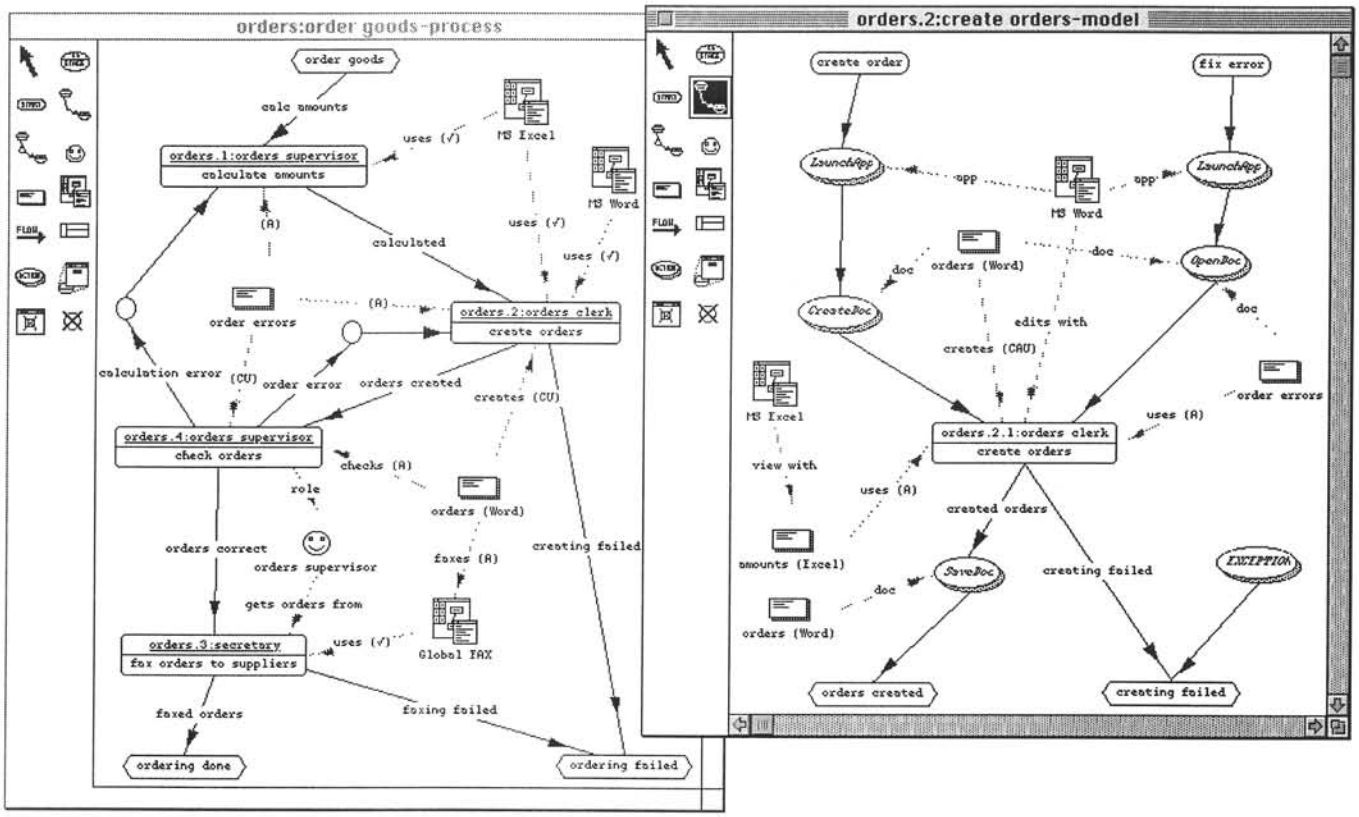

Figure 16. A simple office automation process described in Serendipity.

7. Architecture and Implementation

7.1. MVIEWS

We have developed a framework of object-oriented classes, called MViews, which provides abstractions for implementing ISDEs (Grundy, 1993, Grundy, 1995). New environments are constructed by specialising framework classes to describe the repository and program representation for ISDEs. 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. External tools not built using MViews can be interfaced to the integrated environment by using external views.

MViews supports 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 ISDE environments, including multiple view consistency, inter-component constraints, efficient incremental attribute recalculation, undo/redo, and version control and collaborative facilities (Grundy, 1995).

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 (Grundy, 1994). Snart is a persistent language, with repository and view objects dynamically saved and loaded to a persistent object store.

7.2. INDIVIDUAL MODELLING TOOLS

We have build many ISDEs using MViews. SPE (Snart Programming Environment) was implemented as a stand-alone ISDE for developing Snart programs (Grundy, 1994). It supports analysis, design, implementation, debugging and documentation of object-oriented programs using a variety of graphical and textual views, all of which are kept consistent. MViewsER was implemented as a stand-alone environment for EER and relational schema modelling (Grundy, 1995). MViewsNIAM (Venable,

1995) was implemented as a stand-alone environment for NIAM modelling. MViewsDP (Grundy, 1995) provides graphical dialog painter and textual constraint views for building user interfaces.

We have integrated SPE, MViewsER, MViewsDP and MViewsNIAM to form an ISEE. This was achieved using a technique of integrated, hierarchical repositories (Grundy, 1995, Grundy, 1995). This involves either defining integrated repositories which represent, for example, OOA, NIAM and ER data, or linking individual repository items, such as MViewsER schema fields and MViewsDP form components, with MViews inter-repository relationships. The inter-repository links keep data in different repositories consistent under change in a similar manner to the view relationships described above.

7.3. THE SERENDIPITY ENVIRONMENT

Serendipity itself was built using MViews by specialising MViews classes for representing repository components and relationships, view icons, glue and textual components, and view editors. Constructing EVPL and VEPL diagrams using Serendipity view editors results in construction of repository-level components and relationships describing a process model. Templates are simply process models which can not be enacted.

When enacted, process model component state variables are updated to indicate enactment events. Enactment events are simply represented as MViews change descriptions. These are propagated along event connections, with stages interpreting event change descriptions they receive (AND, OR, and start/stop stages interpret these in a special way). Filters and actions interpret change descriptions by comparing the event change descriptions to a filter pattern, running a Prolog predicate which accesses MViews data (i.e. the API interface), or enacting a subprocess model.

Process stage enactment and modification histories are provided by MViews. Artefact modification histories are constructed as stages receive artefact change descriptions from other MViews environments, or from components which interface to applications not built using MViews and which translate external tool events into MViews change descriptions.

7.4. INTEGRATING SERENDIPITY WITH OTHER ENVIRONMENTS

Figure 17 shows the way Serendipity interfaces with other MViews ISDEs and external tools. If a change is made to a view-level ISDE tool item (1), this is propagated to a repository ("base view") level item change, and the change description generated by this repository-level item change propagated to the ISDE base view component (2). The base view adds artefact, tool and user information to the change description, sends it to the Serendipity environment base view (3), which forwards the change description to the user's current enacted stage (4). This, in turn stores the change in its artefact modification history (5) and forwards it to any filters/actions it is linked to by artefact update event connections (6). The process stage attaches "work context" information to the artefact change description (primarily information about the stage itself), sends it back to the Serendipity base view (7), which returns it to the ISDE tool base view (8). The ISDE base view returns the augmented change description to the work artefact which generated it (9), which stores it in its own modification history (10). View updates are sent to Serendipity in the same manner, with the updated view component sending a change description to the view (11), which forwards it to its base view (12), with the base view forwarding the change description to Serendipity and receiving an augmented change description as before.

External tools, such as MS Word or Excel, can also use Serendipity to provide a process modelling and work context environment.Events from such external tools (Apple Events) are sent to a small translator program (13), which forwards them to Serendipity for processing (14). Serendipity events can be sent to the external tools via the translator (15, 16). These might request the external tool be launched, ask it to save, open or close files, or ask it to carry out some other task (such as send a fax or an email message).

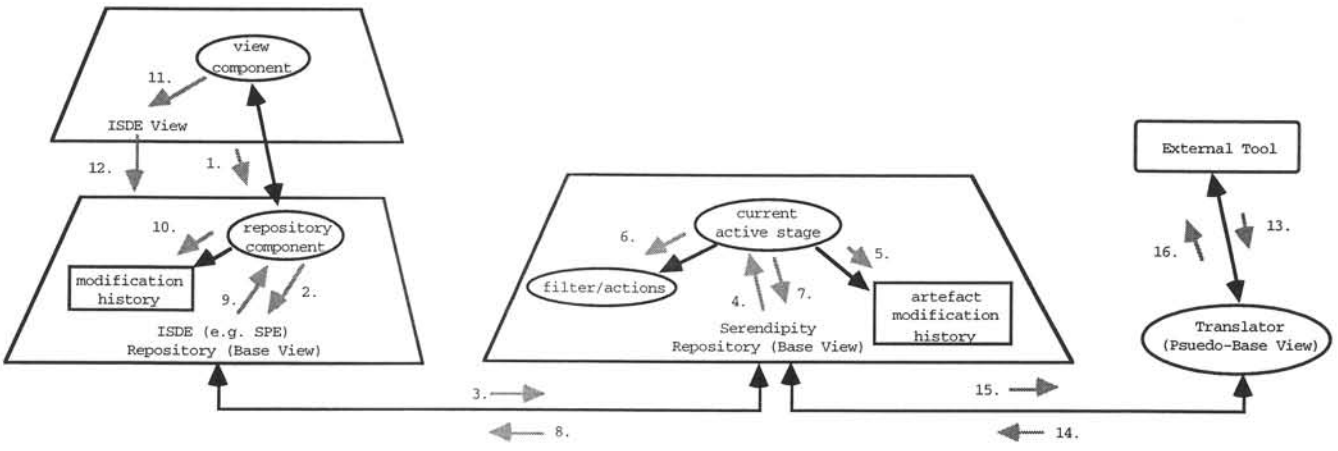

Figure 17. Integrating Serendipity with integrated tool views.

Serendipity's collaborative editing is provided by the C-MViews extensions to MViews (Grundy, 1995). Filters and actions specified on process models work as expected if synchronous editing is used as all collaborators share the same process model data. Asynchronous editing implies collaborators have different process model versions which they edit and enact independently, with other collaborators not being aware of these events. Semi-synchronous editing propagates editing, enactment and artefact update events between the environments of collaborators who express interest in these changes for particular process stages, tools, artefacts or roles. C-MViews base views have been extended to incorporate a notion of *monitors* which detect events of interest to collaborating users. Users request, via the C-MViews server, other collaborators' environments to establish monitors on items of interest. Events are then propagated semi-synchronously, via the server, to the interested user's environment for presentation and/or actioning.

Serendipity performance is generally good, although the C-MViews synchronous editing response time is quite slow (Grundy, 1995). Processing several filters and actions which have been specified on a process stage can also cause performance problems, particularly with hierarchical filters, which are inherited by all subprocess stages. Serendipity stages cache a list of all filters they are linked to, whether directly or hierarchically, as this avoids propagating all enactment and artefact events up the process model hierarchy. We allow users to request filters and actions be actioned semi-synchronously, using the same monitor mechanism used to semi-synchronously propagate events between environments. A user's environment can be instructed to process filters for specified stages in idle time, rather than whenever the stage receives an enactment or artefact event. This greatly improves performance when several filters need to be sent events from the process stage or time-consuming actions must be performed. A drawback is that enforcement strategies are only detected after enactments or artefact changes have been made. Users thus need to accept process model violations occurring and then being resolved after the fact, rather than all checks being made immediately whenever an enactment or artefact change is made.

8. Summary

We have described the Serendipity visual languages and environment which support process modelling, enactment, improvement and work planning and documentation. The Serendipity Extended Visual Planning Language doubles as a notation for process model description and work planning. The Visual Event Processing Language allows EVPL process models to be extended with arbitrary event-handling filters and actions. Serendipity has been implemented using the MViews framework for Integrated Software Development Environments. Serendipity supports multiple views of process models with view consistency, enactable process models, enactment and modification histories, process improvement, reusable process model templates, and metaprocess modelling. It has been used for work context definition, context awareness, collaborative process modelling and work planning for ISEE and office automation applications. Serendipity provides higher-level process model descriptions than most other workflow or PCSEEs, and supports a greater range of work coordination and collaborative work facilities.

Serendipity is being extended to incorporate a flexible Visual Query Language, which will double as a tool repository query and data visualisation language, and a more flexible artefact data specification for use by VEPL filters and actions. Filters and actions are currently interpreted, but will be compiled to Snart, the implementation language of Serendipity and MViews, to improve their performance. We are currently porting MViews to C++, and will also port Serendipity and the MViews suite of software development tools, to improve their accessibility, performance, and ability to integrate existing tools.

References

Apperley, M.D., Gianoutsos, S., Grundy, J.C., Paynter, G., Reeves, S., and Venable, J.R. 1996. A generic, light-weight collaborative notes and messaging facility for groupware applications, Working Paper, Department of Computer Science, University of Waikato.

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

Bogia, D.P. and Kaplan, S.M. 1995. Flexibility and Control for Dynamic Workflows in the wOrlds Environment, *Procs. of the Conference on Organisational Computing Systems*, ACM Press, Milpitas, CA, November 1995.

Ellis, C.A., Gibbs, S.J., and Rein, G.L. 1991. Groupware: Some Issues and Experiences, *Communications of the ACM*, 34 (1): 38-58.

Grundy, J.C. and Hosking, J.G. 1993. A framework for building visusal programming environments, *Procs. of the 1993 IEEE Symposium on Visual Languages*, IEEE CS Press, pp. 220-224.

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

Grundy, J.C., Hosking, J.G., Fenwick, S., and Mugridge, W.B. 1995a. Connecting the pieces, Chapter 11 in*Visual Object-Oriented Programming*. Manning/Prentice-Hall.

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

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

Grundy, J.C. and Venable, J.R. 1995d. Providing Integrated Support for Multiple Development Notations, *Procs. of CAiSE'95,* Finland, Lecture Notes in Computer Science 932, Springer-Verlag, pp. 255-268.

Grundy, J.C., and Venable, J.R. 1995e. Developing CASE tools that support integrated design notations, *Procs. of the 6th European Workshop on Next Generation of CASE Tools,* Finland, pp. 109-116.

Grundy, J.C. and Hosking, J.G. 1996a. Keeping textual and graphical views of information consistent, Working Paper, Department of Computer Science, University of Waikato.

Grundy, J.C. and Venable, J.R. 1996b. Towards an environment supporting integrated Method Engineering, Working Paper, Department of Computer Science, University of Waikato.

Grundy, J.C., Venable, J.R., Hosking, J.G., and Mugridge, W.B. 1996c. Coordinating collaborative work in an integrated Information Systems engineering environment, Working Paper, Department of Computer Science, University of Waikato.

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

Hill, R.D. 1992. The Abstraction-Link-View Paradigm: Using Constraints To Connect User Interfaces to Applications, *Procs. of CHI '92: Human Factors in Computing,* ACM Press, pp. 335-342.

Jaccheri, L., Larsen, J.O., and ., R.C. 1992. Software Process Modeling and Evolution in EPOS, in *Proc. Fourth International Conference on Software Engineering and Knowledge Engineering (SEKE),* Capri, Italy, pp. 17-29.

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

Kaplan, S.M., Tolone, W.J., Bogia, D.P., and Bignoli, C. 1992. Flexible, Active Support for Collaborative Work with ConversationBuilder, in *1992 ACM Conference on Computer-Supported Cooperative Work,* ACM Press, pp. 378-385.

Kellner, M.I., Feiler, P.H., Finkelstein, A., Katayama, T., Osterweil, L.J., Penedo, M.H., and Rombach, H.D. 1990. Software Process Modelling Example Problem, *Procs. of the 6th International Software Process Workshop,* Hokkaido, Japan, IEEE CS Press, 28-31 October.

Kreifelts, T., Hinrichs, E., and Klein, H.K. 1991. Experiences with the Domino Office Procedure System, *Procs. of the Second European Conference on Computer Supported Cooperative Work (ECSCW'91),* pp. 117-130.

Krishnamurthy, B. and Hill, M. 1994. CSCW'94 Workshop to Explore Relationships between Research in Computer Supported Cooperative Work & Software Process, *Procs. of CSCW'94,* ACM Press, April, pp. 34-35.

Lonchamp, J. 1995. CPCE: A Kernel for Building Flexible Collaborative Process-Centred Environments, *Procs. of the 7th Conference on Software Engineering Environments,* Netherlands, IEEE CS Press, pp. 95-105.

Magnusson, B., Asklund, U., and Minör, S. 1993. Fine-grained Revision Control for Collaborative Software Development , *Procs. of the1993 ACM SIGSOFT Conference on Foundations of Software Engineering,* Los Angeles CA, December, pp. 7-10.

Marlin, C., Peuschel, B., McCarthy, M., and Harvey, J. 1993. MultiView-Merlin: An Experiment in Tool Integration, *Procs. of the 6th Conference on Software Engineering Environments,* IEEE CS Press, 1993.

Medina-Mora, R., Winograd, T., Flores, R., and Flores., F. 1992. The Action Workflow Approach to Workflow Management Technology, *Procs. of CSCW'92,* ACM Press, pp. 281-288.

Peuschel, B., Schäfer, W., and Wolf, S. 1992. A knowledge-based software development environment supporting cooperative work, *International Journal of Software Engineering and Knowledge Engineering,* 2 (1), 76-106.

Roseman, M. and Greenberg, S. 1992. Groupkit: A groupware toolkit for building real-time conferencing applications, *Procs. of CSCW'92,* ACM Press, pp. 43-50.

Saeki, M., Iguchi, K., and Wen-yin, K. 1993. A Meta-model for representing software specification and design methods, *Procs. of the IFIP WG8.1 Conference on Information Systems Development,* Como, Italy.

Swenson, K.D. 1993. A Visual Language to Describe Collaborative Work, *Procs. of the 1993 IEEE Symposium on Visual Languages,* IEEE CS Press, pp. 298-303.

Swenson, K.D., Maxwell, R.J., Matsumoto, T., Saghari, B., and Irwin, K. 1994.A Business Process Environment Supporting Collaborative Planning, *Journal of Collaborative Computing*, 1(1).

Tolone, W.J., Kaplan, S.M., and Fitzpatrick, G. 1995. Specifying DynamicSupport for Collaborative Work within wOrlds, *Procs. of the 1995 ACM Conference on Organizational Computing Systems (COOCS '95),* Milpitas, CA, August, pp. 55-65.

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