

Serendipity: integrated environment support for process modelling, enactment and work coordination

JOHN C. GRUNDY¹ AND JOHN G. HOSKING²

¹*Department of Computer Science, University of Waikato, Private Bag 3105, Hamilton, New Zealand*
Email: jgrundy@cs.waikato.ac.nz;

²*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. Support for process modelling and work coordination in such systems also needs to support informal aspects of work which are difficult to codify. Computer-Supported Cooperative Work (CSCW) facilities, such as inter-person communication and collaborative editing, also need to be well-integrated into both process-modelling tools and tools used to perform work. Serendipity is an environment which provides high-level, visual process modelling and event-handling languages, and diverse CSCW capabilities, and which can be integrated with a range of tools to coordinate cooperative work. This paper describes Serendipity's visual languages, support environment, architecture, and implementation, together with experience using the environment and integrating it with other environments.

Keywords. Process modelling, process enactment, process-centered environments, work coordination, environment integration

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 (WFMS) and Process-Centred Environments (PCEs) are examples of tools developed to assist in the construction, use and evolution of formalised process models. While many of these systems support modelling and enacting work processes well, they have some deficiencies. Some use low-level, textual process models which are difficult for end-users to understand and modify (Swenson, 1993; Bogia and Kaplan, 1995), some lack support for inter-person communication (Di Nitto et al., 1995; Ben-Shaul and Kaiser, 1996; Tolone et al., 1995), and many provide either inflexible process models, which inadequately handle exceptions and are difficult to change while in use (Swenson, 1994; Tolone et al., 1995), or do not adequately support the more informal aspects of cooperative work (Kaplan et al., 1996; Tolone et al., 1995).

Computer-Supported Cooperative Work (CSCW) systems provide communication and cooperative editing tools to support collaborative work. These include synchronous editors and tools, such as shared workspaces, telepointers and video conferencing, and asynchronous communication, such as email and version control systems (Ellis et al., 1991). However, many of these systems focus on low-level collaborative editing and/or person-to-person communication issues, and lack support for general work process modelling (Krishnamurthy and Hill, 1995). Some systems, such as ConversationBuilder (Kaplan et al., 1992a), SPADE/ImagineDesk (Di Nitto and Fuggetta, 1995), and Oz (Ben-Shaul and Kaiser, 1994a), attempt to bridge the gap between CSCW tools and process modelling tools but with limited success. Others, such as wOrlds (Bogia and Kaplan, 1995), provide facilities for supporting the informal aspects of work but do not adequately support codified, cooperative process models (Kaplan et al., 1996). Most existing CSCW, WFMS and PCEs tend not to be well integrated with each other, nor with work tools, such as document editors, CASE tools and programming environments (Tolone et al., 1995, Ben-Shaul and Kaiser, 1996). 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. Serendipity provides a novel graphical process modelling language which can be used to represent both general process models and plans for and histories of work. Another graphical language specifies event handling for process model enactment and work artefact modification events. We focus on the design of these languages and a supporting environment for them, together with our experiences in integrating this environment with other systems, including both CSCW tools and tools for performing work, such as CASE, programming environment, and office automation tools. Serendipity provides a “work context” for these tools, and supports high-level work context awareness and work coordination for them. Descriptions of work artefact changes from integrated tools are annotated with work context information 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 used for collaborative process modelling and work planning, process improvement, method engineering and office automation.

2. Problem Domain: Large Cooperative Work Systems

Tools that effectively support collaborative work in large problem domains, where several people, tools and many work artefacts are involved, such as software development or office automation, have a number of important requirements:

- Support for modelling work processes (Barghouti, 1992; Swenson et al., 1994; Tolone et al., 1995). This allows cooperating people to more readily plan and coordinate their work on large problems. Approaches to providing such process modelling range from precise, formal languages (Barghouti, 1992; Ben-Shaul and Kaiser, 1994a; Bandinelli, et al., 1994) to more high-level, graphical workflow languages (Medina-Mora et al., 1992; Swenson et al., 1994; Baldi et al., 1994). Process models should ideally allow work processes to be enforced or used as guidance, and should be readily understandable and modifiable by users.
- The ability to handle arbitrary events relating to process model state changes, work artefact updates or events in work tools (Bogia and Kaplan, 1995; Grundy et al., 1995a). When the state of a process model or work artefact changes, interested collaborators may need to be informed, or automatic responses carried out (Ben-Shaul and Kaiser, 1994a; Bogia and Kaplan, 1995).
- Effective inter-person communication and collaborative editing support is needed (Di Nitto and Fuggetta, 1995; Ben-Shaul and Kaiser, 1996). With process models providing a high level of “work context” information, keeping collaborators aware of each others’ work is necessary (Bogia, 1995). Allowing people to intermittently join or leave cooperative work sessions, or to review the reasons for artefact changes, requires annotated histories of work for process stages.
- Tools for performing work need to be well-integrated with the tools for communicating and editing work artefacts (CSCW tools) and the process modelling tools (WFMS and PCEs) (Di Nitto and Fuggetta, 1995; Ben-Shaul and Kaiser, 1996, Kaplan et al., 1996). This should allow: CSCW tools to be utilised to discuss and annotate both work and process model artefacts; the process modelling tools to control CSCW and work tool behaviour; work performed with the work tools to be readily coordinated by the process modelling tools.

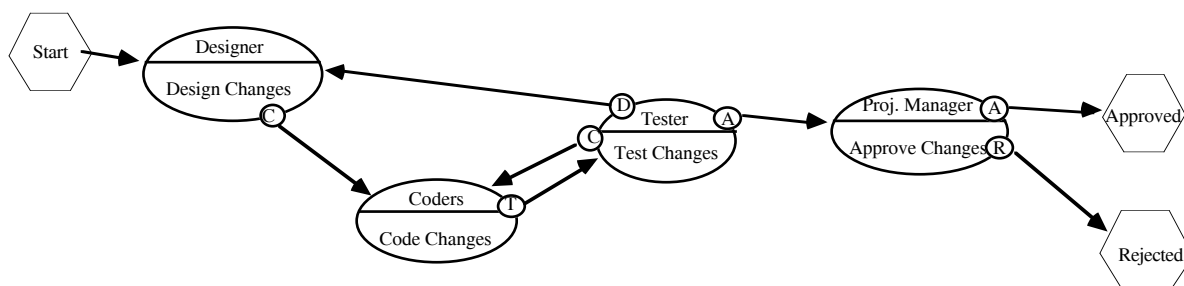


Figure 1. An example VPL process model.

As an example of a process modelling language and environment, consider Swenson's Visual Planning Language (Swenson, 1993), and its embodiment in support environments Regatta (Swenson et al., 1994) and TeamFLOW (TeamWARE, 1996). Figure 1 shows a VPL model describing a process to modify a software system. Ovals are process stages, hexagons start/stop states, and flows between stages represent finishing states of one stage which enact the linked stages.

VPL has many advantages over comparable process modelling languages in that it is concise and versatile. It can be used to both model processes and plan work on specific projects, and Regatta and TeamFLOW allow its process models to be enacted by multiple collaborating users. However, VPL does not adequately model work artefacts or tools, using only simple textual attributes of process stages to indicate this information. It does not support the handling or arbitrary events from related process stages nor tools and artefact updates. The integration of its supporting environments with tools for performing work is limited. For example, TeamFLOW is unable to handle events from work tools or directly control their usage, other than invoking them, when appropriate, from process model stages.

Some work has been done on integrating PCE/WFMS, CSCW and work tools. Examples include ConversationBuilder (Kaplan et al., 1992a), wOrlds (Tolone et al., 1995), SPADE and ImagineDesk (Di Nitto and Fuggetta, 1995), Marvel and ProcessWEAVER (Heineman and Kaiser, 1995), and Oz (Ben-Shaul et al., 1994b; Ben-Shaul and Kaiser, 1996; Ben-Shaul and Kaiser, 1994a). These approaches have produced useful environments with some integration of PCEs/WFMSs, CSCW tools and tools used to perform work. However, this work has so far not produced the "ideal" integration of these technologies, due to problems with adequately integrating disparate tools (Tolone et al., 1995; Valetto and Kaiser, 1995; Di Nitto and Fuggetta, 1995), or with limited scope of the software development and/or workflow tools used (Kaplan et al., 1996, Di Nitto and Fuggetta, 1995).

Serendipity is our approach to solving these problems. It extends VPL to more completely describe work processes, and adds a new visual language for specifying event handling. The next two sections describe Serendipity's visual languages, and are followed by a description of the support environment. Section 6 describes integration of Serendipity with other tools, and is followed by a description of the architecture and implementation. Section 8 describes experience using Serendipity, compares it with other languages and systems, and discusses current and future work.

3. High-level Process Modelling and Work Planning

3.1. EXTENDED VISUAL PLANNING LANGUAGE

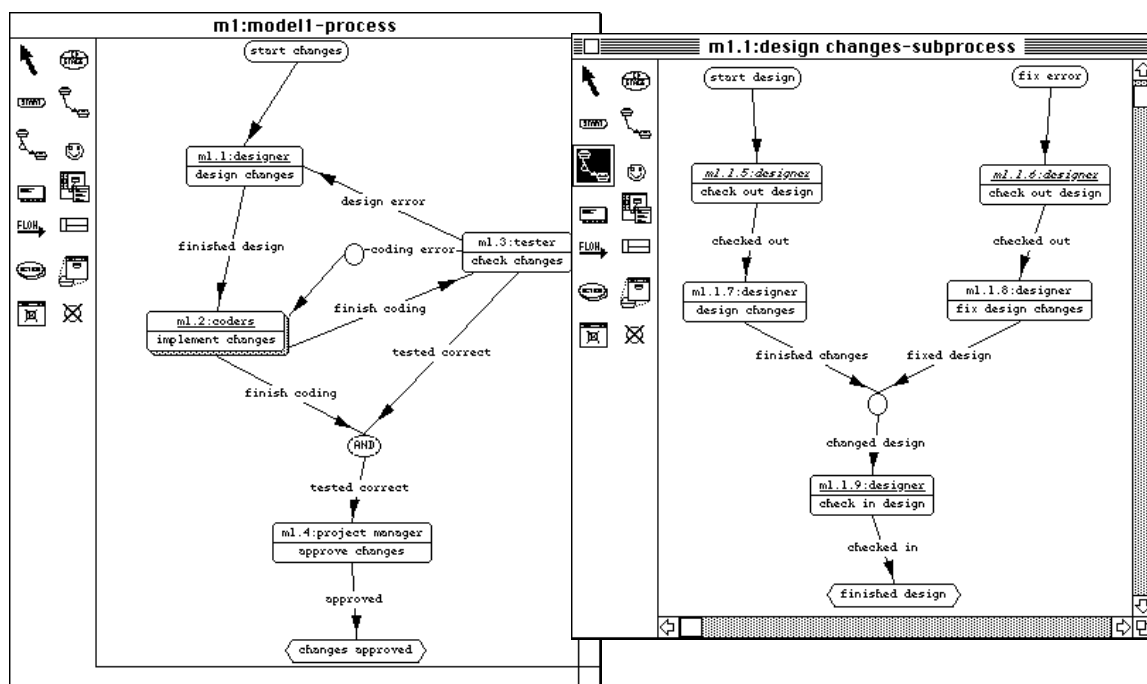


Figure 2. A simple software process model and subprocess model in Serendipity.

We have developed the Extended Visual Planning Language (EVPL), which preserves the notion of simple “work plans” from VPL, but adds capabilities to support general 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. New notational components in EVPL include: identifiers for process stages, which allow stages to be more readily identified and referenced by other process models; role, artefact and tool representations, so these aspects of the work context for a process stage can be specified; “usage” connections between process stages and role, artefact and tool representations; and various annotations on usage connections to indicate how these other parts of a work context are utilised by process stages. Window “m1:modell-process” in Figure 2 uses EVPL to describe a simple software process model for updating a software system. Figure 3 summarises the basic EVPL modelling capabilities.

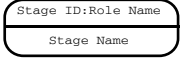
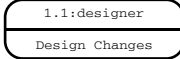
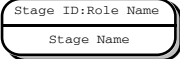
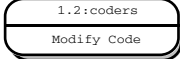
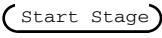
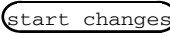

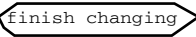
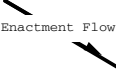
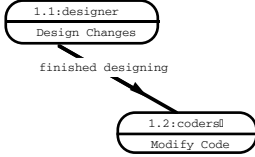

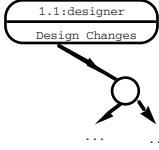

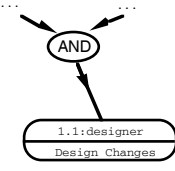
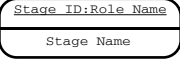
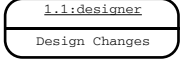
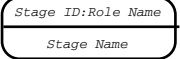
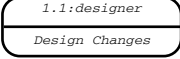
Notational Symbol	Example	Description
		An EVPL Process stage. Stages have a unique ID, a role (person or people who perform the stage tasks), and name. The ID is user-defined and used to uniquely identify stages when they appear in multiple views of a process model.
		An EVPL stage the tasks of which are performed by multiple people (i.e. several people are assigned to the role).
		Start stage for an EVPL diagram. When a process is first enacted, an enactment flow event will flow in from one of its start stages and onto the connected process stage(s).
		Stop stage for an EVPL diagram. When a finishing stage flows into a stop stage, the EVPL process has completed.
		An enactment event flow from one stage to another. The label on the enactment flow is usually an indication of the finishing state of the process stage from which the flow is from. In the example, if stage 1.1:Design Changes finishes in state “finished designing”, stage 1.2:Modify Code is enacted.
 (OR stage)		An OR stage. When an enactment event flows into an OR stage, the event flows into all stages connected to the OR stage. This can be represented by multiple enactment flows from one stage to several others, but using an OR stage often makes diagrams more readable and more clearly indicates that the connected stages can be enacted simultaneously. In the example, when 1.1:Design Changes completes, the two stages the OR flows into are both enacted.
 (AND stage)		An AND stage. If two or more enactment flows from different stages flow into an AND stage, the AND stage ensures that the stage(s) into which it has enactment event flows is not enacted until all stages flowing into the AND complete. In the example, 1.1:Design Changes will not be enacted until both stages flowing into the AND complete.
		Underlining of a stage ID and role indicates the presence of one or more expanded subprocess for the stage i.e. the tasks to perform for the stage are described on one or more expanded EVPL diagrams.
		Italicising the stage ID, role and name indicates the stage has been instantiated from a reusable template. Any subprocesses defined for the template are also copied and made subprocesses of the instantiated stage.

Figure 3. Basic modelling capabilities of EVPL.

The Figure 2 example would normally be part of a larger process, such as the ISPW-6 software process example (Kellner et al., 1990). Several stages are shown, each describing part of the overall process. Process stages “m1.1: design changes” and “m1.2: implement changes” have subprocess models. The expanded subprocess model for “m1.1: design changes” is shown in the window on the right. Stages m1.5 and m1.6 in the subprocess model have been instantiated from templates.

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) connecting m1.1.7, m1.1.8 and m1.1.9, indicates m1.1.9 is enacted when either m1.1.7 or m1.1.8 finish.

To use this process model, stages in the model are *enacted*. When a stage completes in a given state, event flows with that state name (or no name) activate to enact linked stages; enacted stages and the enactment event flows causing their enactment are highlighted. 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. that user's work context). Section 5 further describes process enactment in Serendipity.

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 particular project the process is used on. 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 indicates work plans are defined for each "coder" for stage m1.2. The people filling these "coder" roles can define or modify these work plans or have them defined by some other role (perhaps the "designer" or "project manager").

Figure 4 shows the EVPL work plan definition for the "m1.2: implement changes" stage for coder "john". When this stage is enacted with "start coding", "john" is to 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", "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 there is a mapping from the "finished design" and "coding error" finishing states of m1.1 and m1.3 to the "start coding" and "fix code" starting state names of the subprocess model. This mapping is specified in a dialog when adding each enactment event flow. Stages can be enacted in the same start state by multiple event flows, such as stage m1.2.6 in Figure 4.

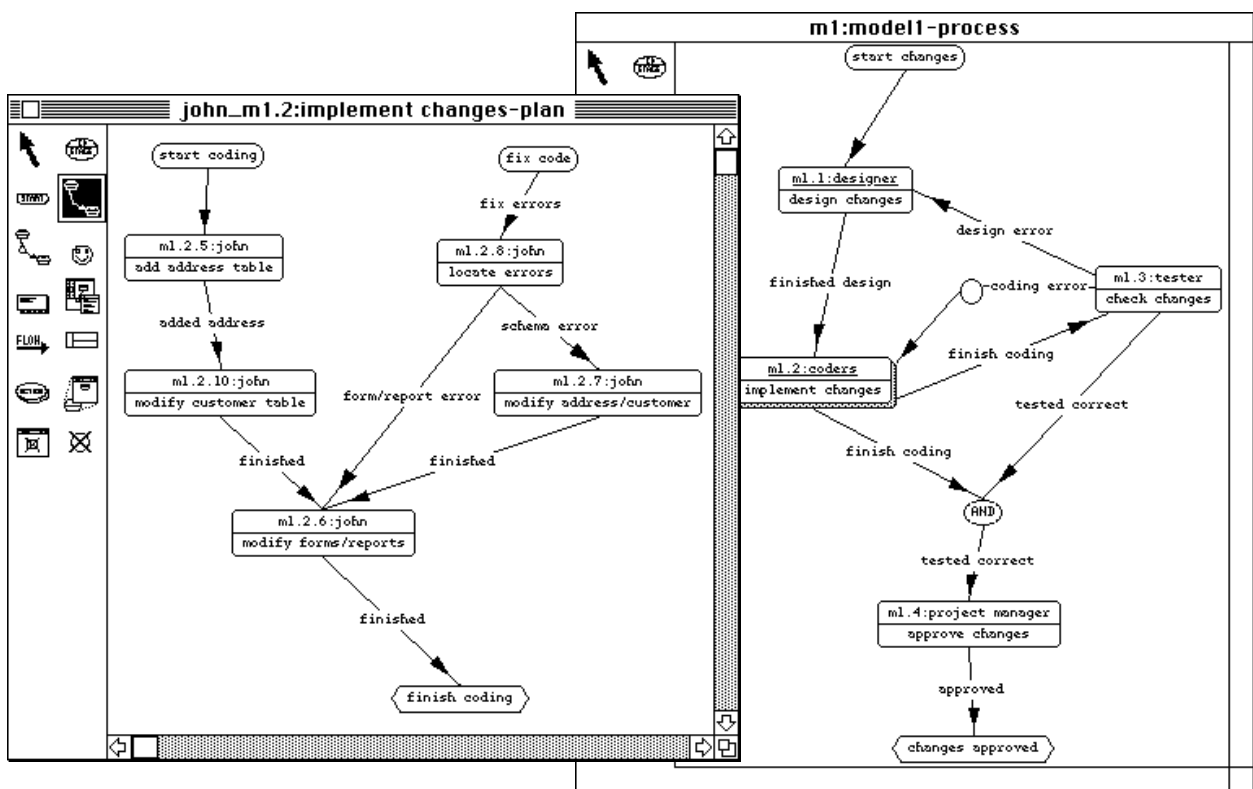


Figure 4. A work plan for coder "john".

4.2. DESCRIBING PROCESS STAGE WORK CONTEXT INFORMATION

EVPL also extends VPL to permit capture of *work context* information in the form of tools and artefact used in each stage, and coordination and communication needed between stage roles (Figure 5). Some, but not all, of these additional modelling capabilities are found in process modelling languages developed by other researchers (for examp, E³ PML (Baldi et al., 1994)).

Figure 6 shows a different perspective (view) of the process model of Figure 2. This retains the process stages, but not the enactment event flows. Instead, usage flows describe the tools, artefacts and roles that the 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” specifies the latter is used to modify the former. The usage connections to the “designer” role from m1.1 and m1.3 indicate coordination between the m1.3 role (“tester”) and the m1.1 role (“designer”), in this case the roles must communicate informally (“talks with”). Annotations on the usage flows indicate, for example, that the designer must use the ER modeller to design changes (√), the ER design artefact is updated by m1.1:design changes (U), the changes list artefact is created and/or updated by m1.1:design changes (CU), m1.2:implement changes accesses (but doesn't update) the ER design (A), and process stages m1.2 and m1.3 cannot be enacted at the same time as process stage m1.4 (↯).







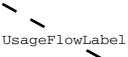

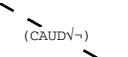

Notational Symbol	Example	Description
 artefact	 class	These describe the kinds of work artefacts used by process stages. Artefacts can represent a general class of work artefact (e.g. “classes” in an OO system), or specific instances of artefacts (e.g. “class ‘window’”).
 ToolName	 ER Modeller	Tools are used to modify or view work artefacts. If a meta-process model is being defined, tools may include the Serendipity environment itself, and artefacts may include process model components.
 RoleName	 john	Roles represent people or abstract roles associated with process stages, tools or artefacts. Roles may refer to specific people (e.g. “john”), groups of people (“coders”), or abstract roles (e.g. “designer”).
 UsageFlowLabel		Usage connections indicate that an artefact, tool, role or process stage is used by another. The flow is from the used process model component to the using process model component.
 (CAUD√-)		Various annotations can be added to a usage flow: ‘C’ = using process model creates instances of used component; ‘A’ = using component accesses used component; ‘U’ = using component updates used component; and ‘D’ = using component deletes instances of used component. C, U and D are generally only useful when the used component is an artefact. √ specifies the using component must use the used component, whereas ↯ specifies the using component must not use the used component. For process stages, ↯ specifies non-concurrent enactment, √ concurrent enactment.

Figure 5. Extra modelling capabilities of EVPL to describe work contexts.

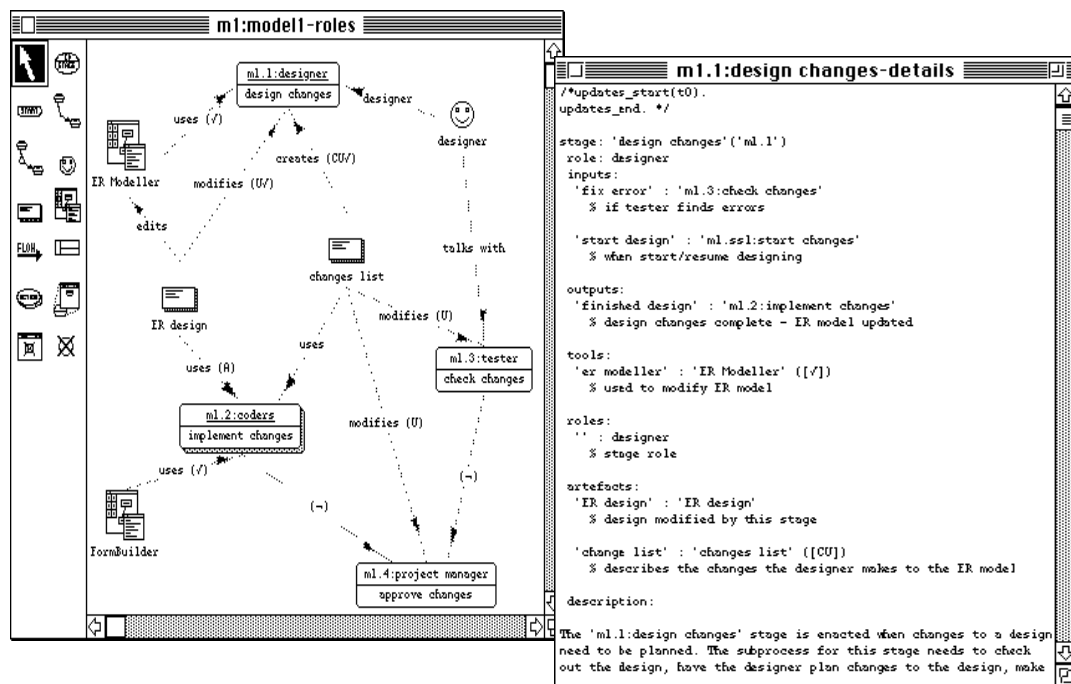


Figure 6. A data, tool and role-oriented perspective of the first process model (left) and a textual stage view (right).

3.3. TEXTUAL PROCESS INFORMATION

The graphical process model, work plan and work context views described above, can be supplemented using textual representations of process stage, artefact, tool and role information. An example is given in Figure 6. Extra information which can be specified includes: user-defined attributes for any kind of process model component (for example, a “percent_complete” value for a process stage); user-defined comments about process model component information; and bindings of abstract role to concrete role (e.g. that “john” will fill the “designer” role).

4. Defining Event-Handling Filters and Actions

The process models described in the previous section are fairly static. Stage enactment flows are described, which indicate the flow of enactment events between stages. Often, however, other types of event handling are required, for example to specify: dependencies between process stages which belong to different process model views; the handling of artefact update and/or tool events; automatically-applied rules or constraints on process models, driven by event triggers; and the automatic invocation of inter-person communication tools.

Few graphical process modelling languages support such capabilities. Most typically use some form of textual specification of process model stage attributes to specify limited forms of event handling, or utilise complex rule-based languages. Neither of these approaches capture and represent graphically the kinds of events nor how they are handled. To this end we developed the Visual Event Processing Language (VEPL), to permit visual specification of arbitrary event handling and event-triggered process model rules.

4.1. BASIC EVENT-HANDLING

The basic VEPL constructs are *filters* and *actions*, which receive *events* from stages, artefacts, tools or roles, or other filters and actions. Filters match received events against user-specified criteria, passing them onto connected filters and actions if the match succeeds. When actions receive an event they carry out one or more operations in response to the event. These operations update information and/or generate new events, which may be detected and acted upon by other filters and actions. VEPL is fully integrated with EVPL and thus provides a graphical, high-level specification of event handling for EVPL process models and work plans.

Figure 7 shows the main components of VEPL. Some of these, such as enactment event flows and usage connections, are derived from EVPL, but have different semantics when used with VEPL filters and actions.

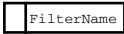
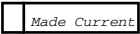


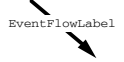
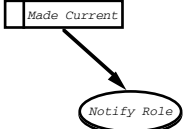
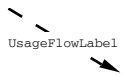
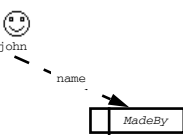
Notational Symbol	Example	Description
		A filter definition. Filters receive events (from process stages, artefacts, tools, roles, other filters or actions) and if the event matches the defined selection criteria for the filter, the event is passed onto the connected filters and/or actions. A filter or action reused from a template filter/action definition has its name in italics.
		An action definition. Actions receive events (from process stages, artefacts, tools, roles, filters or other actions) and respond to the event by performing some action (which often generates other events). Actions can pass on events to other filters and/or actions.
		An event flow into/from a filter or action. Events may be process stage enactment events, artefact update events, tool events, some event caused by a role (i.e. user), or an event generated by an action. For example, if Made Current decides an enactment event flowing into it means a process stage has been made the current enacted stage, then the Notify Role action is invoked to notify another user about this event.
		Usage flow into a filter or action. These specify parameters of the filter/action. For example, the MadeBy filter is parameterised by a role name which it uses to decide whether some event was caused by a particular role. In the example, that role name is instantiated to "john" by the usage connection to the role process model component.

Figure 7. Basic modelling capabilities of VEPL.

Figure 8 shows a simple event-handling view which extends the process model from Figure 2, illustrating use of VEPL for inter-stage work coordination. In this example, testing of software (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 8 specifies when notifications are sent to coders that testing has been completed or is in progress.

The lefthand event flow from m1.3 specifies that all enactment events on m1.3 should flow into the filter “Made Current”. This checks if the received event shows the stage 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” is parameterised by a stage id, instantiated in this case to stage m1.2. This filter checks its stage parameter is either enacted or able to be restarted (not yet completed) and, if so, passes received event flows into action “Notify Role”. This action, parameterised by a role, informs the person or people filling that role (in this example, the “coders” of m1.2), that testing has started or finished. Notification is, by default, via an e-mail like message, but users can specify, via dialogue box, they wish to be informed in other ways, such as by opening a dialog on the coders’ screen, shading the “m1.2:implement changes” stage icon to indicate presence of a message, etc (Grundy et al., 1996a). 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.

“Notify Role”, “Made Current” and “Not Complete” are reusable (system defined) filters/actions while “finished testing” is specific to this particular process model. Filters are defined by a pattern-matching language (for simple filters), instantiation of a reusable template (via a forms interface), or use of an API to Sernipity’s implementation language (see later in this section).

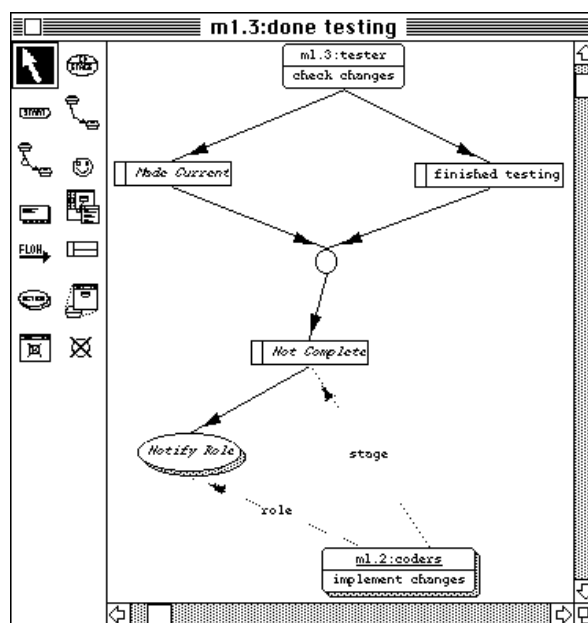


Figure 8. An example of process stage event filtering.

4.2. HANDLING ARTEFACT, HIERARCHICAL AND MULTIPLE EVENTS

To allow different processing based on the kinds **or** numbers of events, VEPL allows users to: distinguish between enactment and tool/artefact update events; specify if subprocess component events are handled; and specify that a filter/action is to handle sequences of, rather than individual, events. Figure 9 shows these additional modelling capabilities.

Notational Symbol	Example	Description
		The filter or action is informed of any artefact update (or tool/role) events, rather than enactment events. In the example, FilterName is informed whenever an artefact update event occurs and process stage 1.1 is the current enacted stage (i.e. the stage on which the user is currently working). If the \mathcal{A} was not present, the filter is only be informed whenever stage 1.1. is enacted or de-enacted.
		The filter or action is informed when stage 1.1. <i>or any of its subprocess stages</i> are enacted or de-enacted. In the example, FilterName is informed whenever an enactment event occurs for stage 1.1 or any of its subprocess stages. If \mathcal{A} and Σ are combined, the filter is informed of any artefact update events made when stage 1.1. or any of its subprocess stages are the current enacted stage.
		The filter or action is informed of a series of events from the connected process stage (or artefact, tool or role), as opposed to being informed of single events. In the example, several enactment or deenactment events may occur, all of which are sent to FilterName and it will decide on the action it will take based on this sequence of events, not just as each event is received. The * is often combined with Σ and \mathcal{A} annotations.

Figure 9. Extended modelling capabilities of VEPL.

As an example, figure 10 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 (indicated by the \mathcal{A}) made by any user filling the “coder” role while working on m1.2, or any of its subprocesses (indicated by the Σ), flow into a filter (“RDB table change”), which checks if the change is to a table. If so, the “designer” is notified of the event. “RDB table change” can be defined in two ways: as 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.

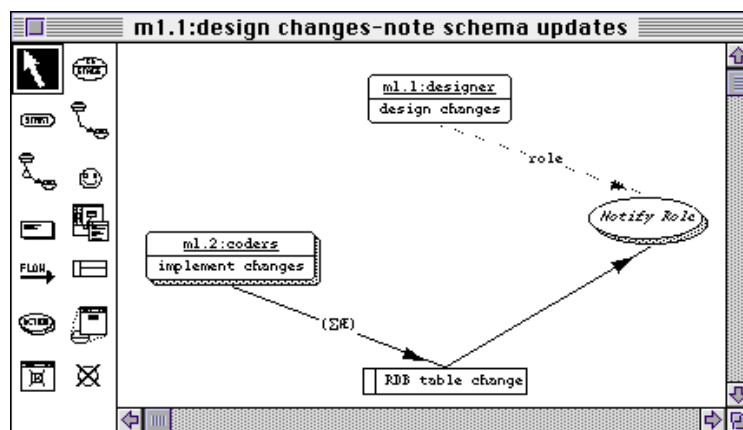


Figure 10. An example of artefact change event filtering.

Roles, tools and artefacts used in a VEPL specification act as filters, if events flow into them from other process model components. For example, Figure 11 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" owned by “rick”. Artefact update events can also be filtered through roles and tools. The m1.2 stage can be removed from the example in figure 11, and the flow from artefact “customer table” annotated with \mathcal{A} . 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.

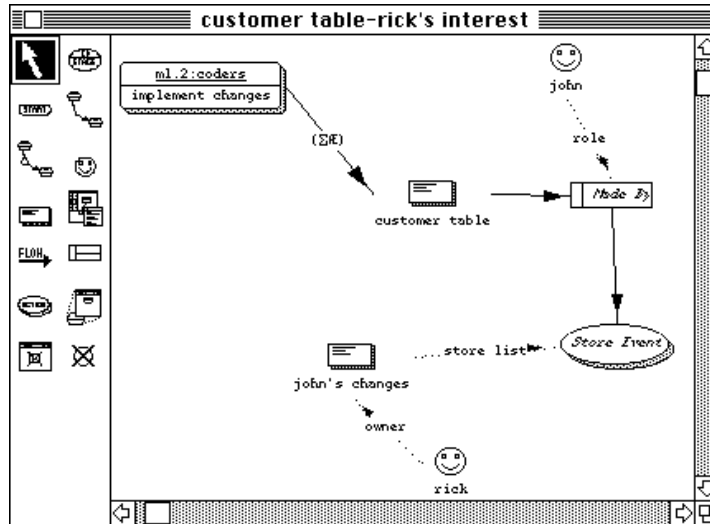


Figure 11. An example of complex filtering and actions.

Event flows annotated with a * indicate multiple events from a stage, artefact, tool or role are to be processed. The attached filter/actions receive multiple events before deactivation, allowing them to recognise and process complex event sequences, and maintain state between the individual events. They can thus act as semi-autonomous agents. Such filter/actions reset their state after receiving the sequence of events they are interested in. For example, in Figure 12 the “Add Entity & Rel/Roles” filter recognises a sequence of ER updates (addition of a new entity, relationship, roles and attributes) and notifies coders of the entire table update using a single abstract message, rather than via a long list of messages describing each discrete change made. In this case, the filter accepts ER modeller artefact update events until it recognises an unrelated artefact (e.g. a different table) is being modified. It then passes on a single event to “Notify Schema Affected” and resets its state.

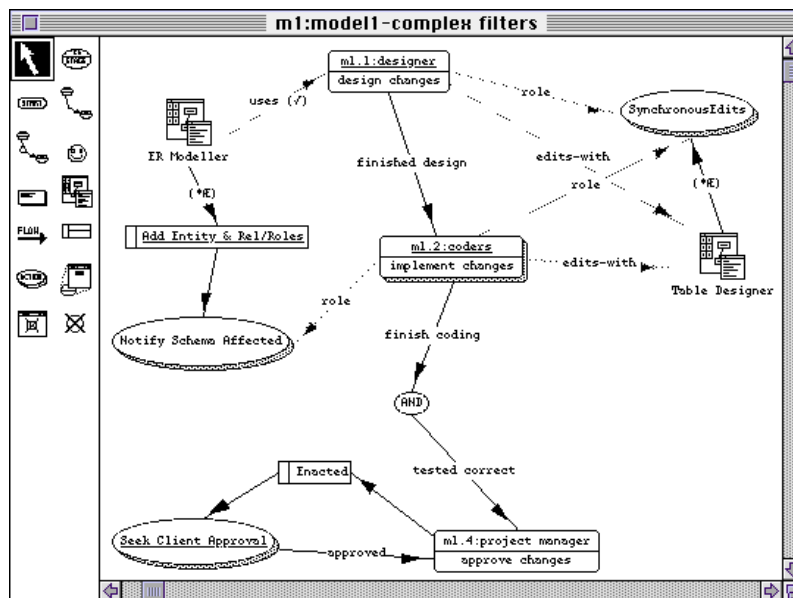


Figure 12. Multiple event processing, stage coordination and external process interfacing.

Another example of a complex filter/action is the “SynchronousEdits” action, used in figure 12 to specify how work is to be coordinated between people filling the roles for two concurrently enacted stages. In this case, if the m1.1 and m1.2 stages are concurrently enacted and the roles involved both use the “RDB Table Designer” tool, they must use the synchronous editing mode of that tool to ensure schema changes they are making do not conflict. The “Seek Client Approval” action models a non-computerised process, asking the “project manager” to liaise with the customer and gain approval for the new system changes made.

4.3. DEFINING FILTERS AND ACTIONS

Template Filter/Action	Kind	Parameters	Description
Enacted	filter		On receipt of an event, checks if this event is a "start" (i.e. stage enactment) event.
Deenacted	filter		On receipt of an event, checks if this event is a "finish" (i.e. stage deenactment) event.
Made Current	filter		On receipt of an event, checks if this event is a "made_current" (i.e. stage made the current enacted stage) event.
Finished Current	filter		On receipt of an event, checks if this event is a "finished_current" (i.e. stage finished being the current enacted stage) event.
Not Complete	filter	stage	On receipt of an event, passes on event if the given stage has not been completed (i.e. stage's "completed" flag not set).
Complete	filter	stage	On receipt of an event, passes on event if the given stage has been completed (i.e. stage's "completed" flag is set).
Made By	filter	role	On receipt of an event, checks the event was caused by the specified role(s).
Edited By	filter	tool	On receipt of an artefact update event, checks the event was caused by editing using the specified tool(s).
Notify Role	action	role	On receipt of an event, notifies the people filling the role of the connected process stage(s) that the event has occurred.
Store Event	action	update list	Stores the event received in an "updates list" artefact (a list of event record descriptions).
Wait	action		On receipt of event, queues the event and does not forward it to connected stages, filters or actions until the user-specified period to wait has completed. Multiple events can be queued using the * annotation on the event flow.
LaunchApp	action	tool	Launches the specified tool.
CreateDoc	action	tool, artefact	Instructs the specified tool to create a new artefact of the specified type & name.
OpenDoc	action	tool, artefact	Launches the specified tool (if not already running) and instructs it to open the specified artefact(s).
SaveDoc	action	tool, artefact	Instructs the specified tool to save the specified artefact(s).
CloseDoc	action	tool, artefact	Instructs the specified tool to close (i.e. stop working with) the specified artefact.
QuitApp	action	tool	Instructs the specified tool to quit.

Figure 13. Some of the template filters and actions provided by Serendipity.

Some of the filters and actions shown in the preceding examples are reusable library templates, some are user-defined by simple specification of patterns to match, some are defined by subprocess models, and others utilise an API interface to the implementation language of Serendipity. Examples of commonly-used template filters and actions are shown in Figure 13. Users can extend the library by adding their own filters and actions.

VEPL filter/actions can be defined using VEPL itself by expanding a subprocess, parameterised by start and finish stages, similar to EVPL stage subprocesses. The filter or action icon is then connected to appropriate process model components, and when it receives an event its subprocess is used to determine the response to the event. Figure 14 is a generic filter/action "update data model", which checks if a change to a given "entity" artefact is an entity rename or attribute modification (add, delete, rename or change type), and, if so, stores the artefact change event in an "updates list" specified. The finish stage "update" represents an output from this filter/action, allowing the event to be flowed onto other filters/actions. This filter/action is reused by adding an action icon "update data model" to a process model view. The user then specifies an event flow from an entity artefact into "update data model", and a usage flow into "update data model" from an "updates list" artefact. The event flow out of "update data model" can optionally be used, as required.

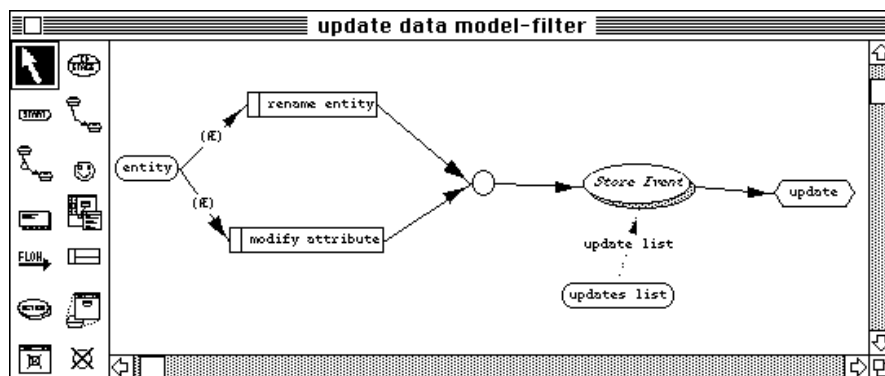


Figure 14. A parameterised filter/action.

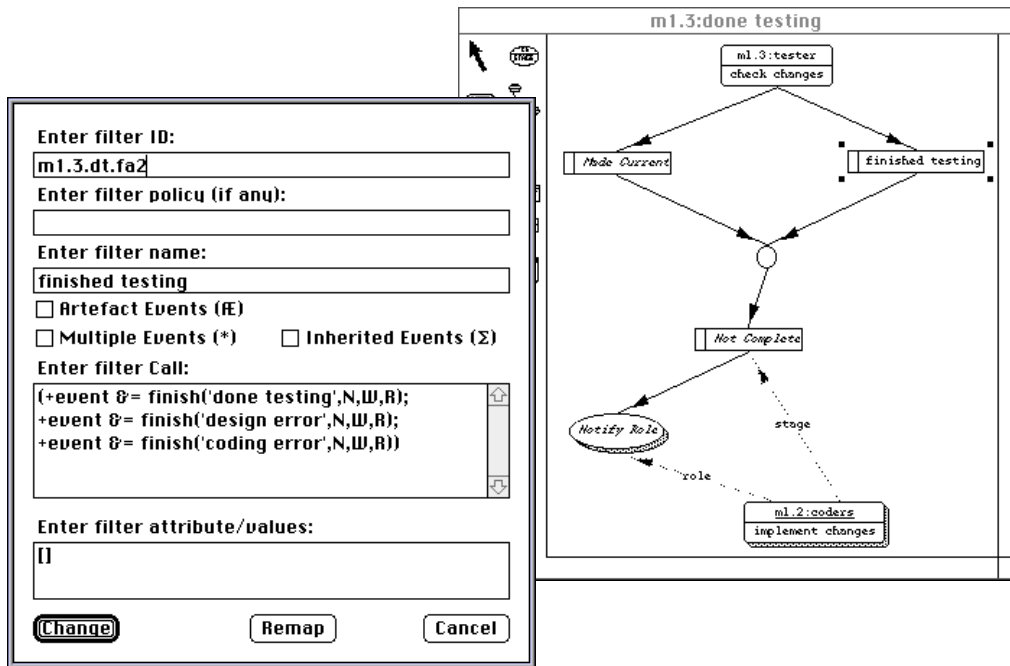


Figure 15. Example of specifying detailed filter and action information.

Filters and actions can also be implemented via a Prolog-based API to Serendipity's implementation. Pattern-matching on input event representations is used to differentiate between different event types. Using the API, filters or actions may call internal functions or access the internal data structures of Serendipity to implement complex filtering operations, or actions.

Figure 15 shows the specification of filter “finished testing”. Names and ids are specified in the top three fields. Flags are set to specify the types of event the filter is interested in. The filter call field is either blank (if the filter action is copied from a template or has a subprocess model), or contains Prolog calls. The “&=“ operator used in “finished testing” compares an enactment event received by the filter (indicated by “+event”) with the given pattern. In this example, the filter responds to finish events in the states 'done testing' 'design error', or 'coding error'. Various other operators and values can be used when specifying the filter/action API call(s), such as the \$= operator which compares artefact events to a pattern, the +self value (the object ID of the filter/action receiving the event), and the +from value (the object ID of the model component the event is from). Any parameter inputs to the filter/action specified by usage connections are referred to by +UsageFlowName, where UsageFlowName is specified in a dialog for the usage connection when it is added.

The bottom edit box (“Enter filter attribute/values:”) is used to specify state information for a filter/action. This is usually used to constrain or modify filter/action behaviour. For example, for the “Notify Role” action, this has the value “(notify_method=message)”, specifying that for this instantiation of the Notify Role template action, coders should be notified by a message (using an email-like messaging system described in Section 5). Other values of this notify_method attribute include: “open_dialog”, to notify users via a dialogue; “highlight”, via highlighting a process stage icon; or “broadcast”, via a broadcast message appearing at the bottom of the other users’ screens.

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 task itself.

5.1. MULTIPLE PROCESS MODEL VIEWS

EVPL and VEPL process model descriptions are produced using the same editing tool. Several views of the same process model can be defined, each adding more information about the model as a whole. For

example, Figure 3 is a process model for software system enhancement, Figure 6 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 (Figure 6). Serendipity keeps all views of a process model consistent under change, or at least, in a known state of inconsistency (Grundy et al., 1995a).

5.2. PROCESS ENACTMENT

A Serendipity process model can be enacted by users for a particular project at any time. 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.

A process model is enacted by selecting the stage to enact and selecting a menu option. The user also specifies a reason for the enactment and, if the stage has subprocess models defined, a start stage to commence in. An enactment event is then sent to the process stage, and start stage (if specified). The start stage immediately sends enactment events to all stages connected to it. As work on a stage is completed or a subprocess completes (a finish event flows into a finish stage), finish 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. Users can manually complete enacted stages (if no actions prevent this) by supplying a finish state for them.

Figure 16 shows an enacted process model. Enacted stages (ie those started and not yet finished) are shaded. Event flow(s) which have activated a stage are also highlighted. The user's current enacted stage (the stage that user is currently working on) is bold highlighted. Users can select any enacted stage they fill the role of to be their current enacted stage. Users can view a list of their enacted stages, also shown in the figure, and this can function as a "to-do" list for that user.

When a stage is enacted, finished, or made/unmade current, it records this event in an enactment history. A reason is also stored specifying 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 16 illustrates these two histories for stage m1.1.

"EXCEPTION" actions can be defined for process models. These have no input event flows, but are automatically invoked 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. This usually indicates an error in the process model. "EXCEPTION" actions reduce the complexity of process models by eliminating many error-handling actions and event flows. Multiple "EXCEPTION" actions can be defined for a process model with filters used to determine which one handles a particular exception. If not defined for a model a default handler informs 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.

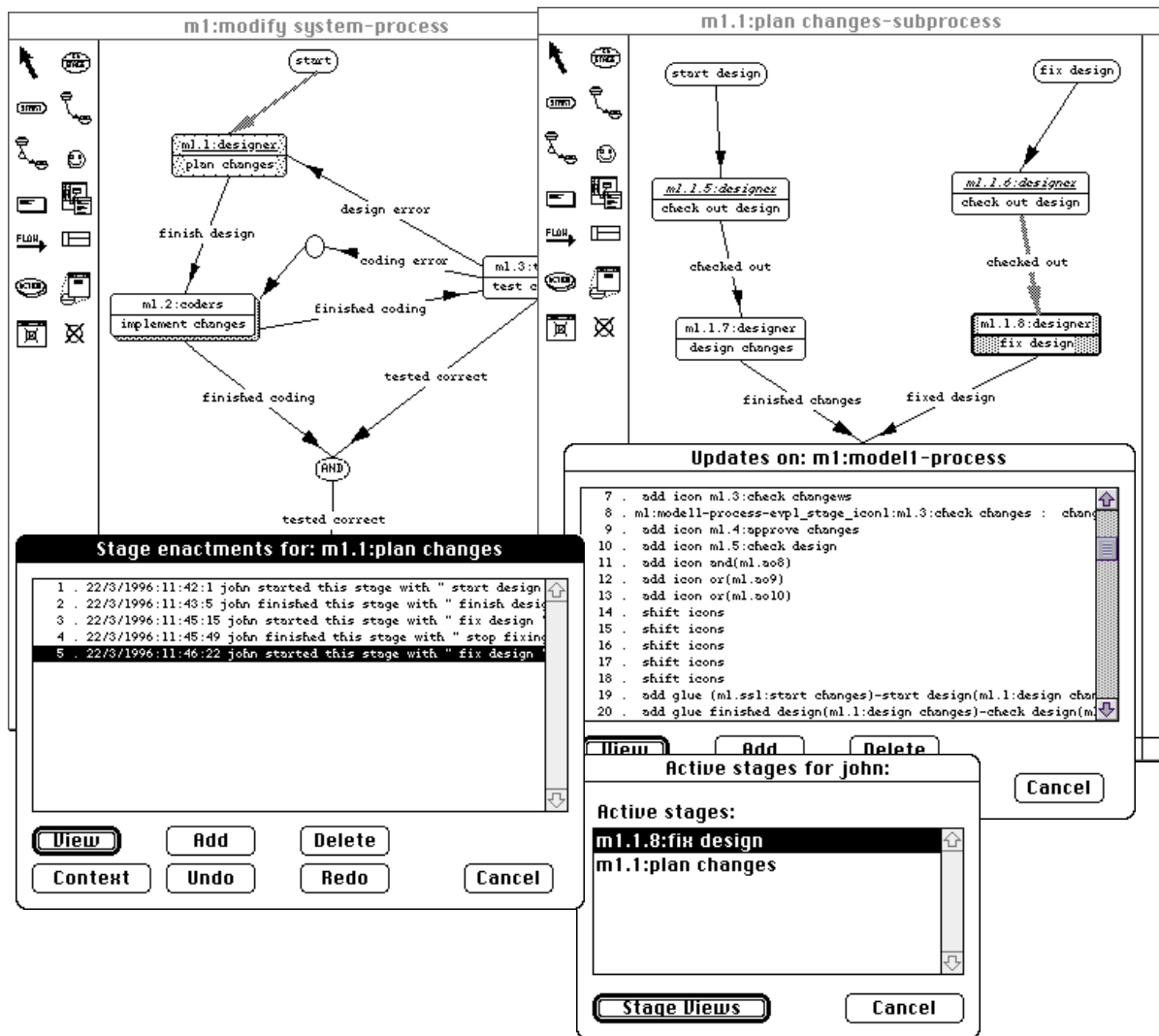


Figure 16. Enactment and modification histories for a stage, and a user's to-do list.

5.3. COLLABORATIVE PROCESS MODELLING AND WORK PLANNING

Serendipity process model views may be shared amongst developers and synchronously, semi-synchronously or asynchronously edited. Serendipity has also been integrated with several small CSCW tools which provide work context-dependent notes, messaging and text chats to facilitate inter-person communication (Grundy et al., 1996b).

To collaborate effectively on a large project, users need higher-level awareness support in addition to the low-level messaging, communication and view editing mechanisms (Grundy et al., 1996b). Serendipity provides information about the work contexts' of their collaborators in a high-level, synchronous way as shown in Figure 17, by highlighting and colouring collaborators enacted stages. Actions can also be specified to highlight artefacts currently being modified by other users, such as the "changes list" in Figure 17. Users may have more than one EVPL view open, with enacted and current enacted stages highlighted.

5.4. PROCESS IMPROVEMENT, REUSE AND META-PROCESS MODELS

Process model often need modification during or after use, as exceptions or unforeseen events arise, and may evolve as users become more proficient at describing their work processes, or because details of a process used varies between projects. Serendipity allows process models to be modified at any time: before, during or after use.

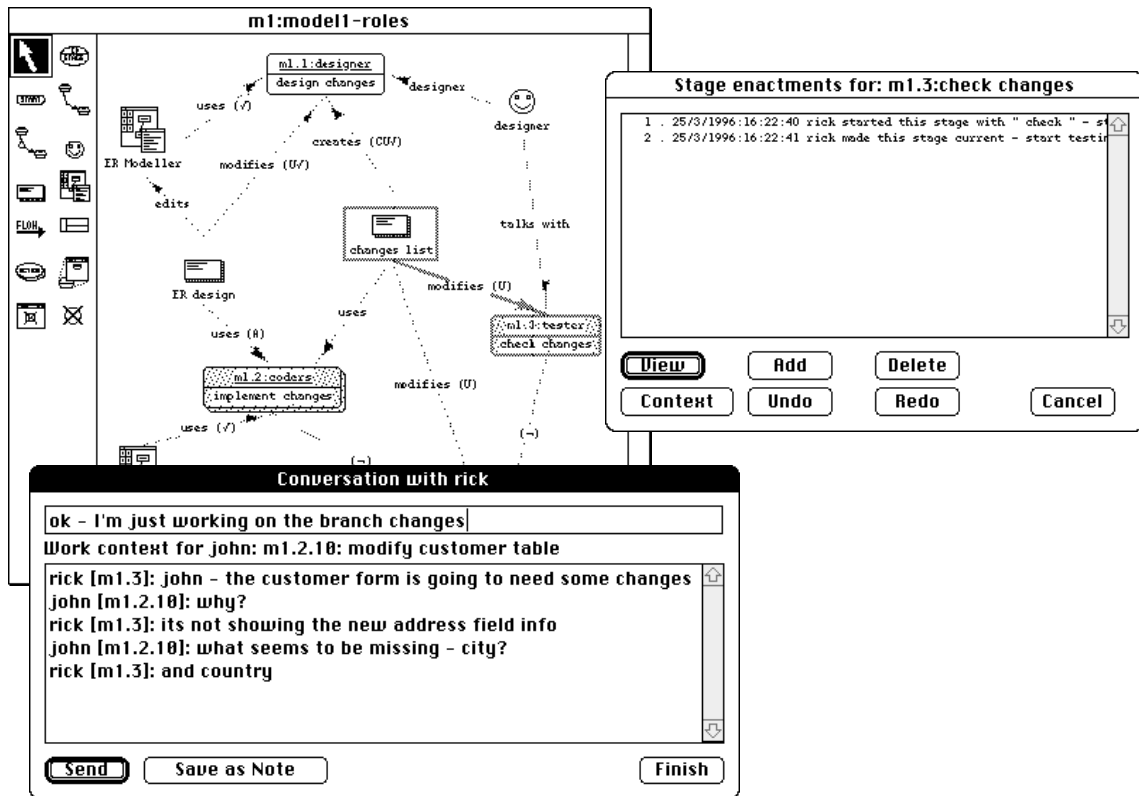


Figure 17. An example of highlighting another user's current process stage and artefacts.

To aid in model evolution, Serendipity can provide statistics about the complexity (number of substages, 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 (Grundy et al., 1996b).

Process stages or filter/actions can be abstracted into process templates. Users can view and edit template models but templates cannot, however, be enacted. Templates are abstract and must first be instantiated (copied) by adding a stage to a process model, specifying it is based on a template.

To guide the process modelling, enactment and improvement process itself, meta-process models can be defined as EVPL and VEPL models. These are also useful for coordinating the modelling 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 themselves, 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 provides more than a process modelling and management environment. Used with other tools it specifies the *context* of work for multiple collaborators, facilitating activities such as collaborative software development, method engineering and office automation.

A user's current enacted stage defines the work context for that user. Any work artefact changes made using other tools are assumed by Serendipity to be part of the work associated with this stage. Changing the current enacted stage changes the context of work. To illustrate the benefits of this, we briefly describe how Serendipity provides a process modelling tool for an integrated Information Systems Engineering Environment (ISEE) (Grundy et al., 1996c). This environment includes the SPE object-oriented software development environment (Grundy et al., 1995b), an ER modelling tool (MViewsER) (Grundy and

Venable, 1995a), a form/report building tool (MViewsDP) (Grundy et al., 1996d), and a NIAM modelling tool (MViewsNIAM) (Venable and Grundy, 1995).

Serendipity does more than just define the work context for a user. As shown in Section 5, stages record their enactment and modification histories. They also record the work artefact updates made while the stage is the current enacted stage, thus documenting the history of work for this work context. Figure 18 shows an example of changes made to an MViewsER view stored for the “m1.2.10:modify customer table” stage for coder “john”.

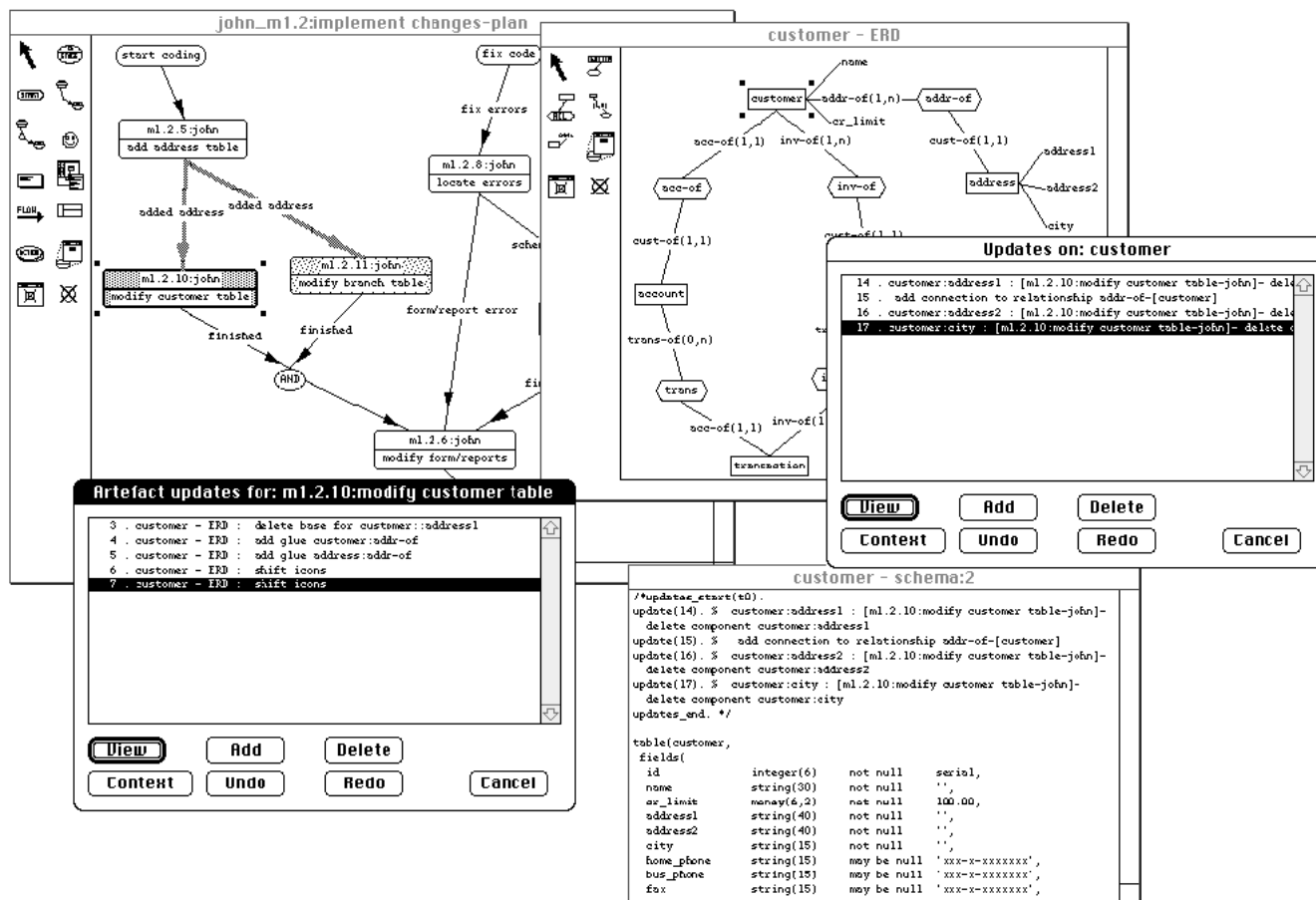


Figure 18. A work artefact change history in Serendipity and work artefact modification history in MViewsER.

In addition to recording work artefact updates in Serendipity stages, work artefact change descriptions stored by the tools themselves are augmented with information about the work context they were made in. Users of these tools can review work artefact modification histories to determine the changes made, who made them, and when and why they were made (i.e. the work context they were made in). Figure 18 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.

While Serendipity highlights process stages, usage connections and artefact and tool icons in use by another developer, as shown in Figure 17, it also allows actions to be specified which highlight work artefacts in integrated tool views (Grundy et al., 1996b). Filter/actions defined by a developer can detect work artefact changes made in an integrated tool by someone else, and send messages to the tool of the developer to highlight these icons. Due to this degree of integration of Serendipity and our ISEE, users can also 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.

Serendipity actions can be used to constrain some of the integrated tools' functionality, facilitating "Method Engineering" within our integrated environment. 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 and Brinkkemper, 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 and Brinkkemper, 1995). We have used Serendipity to support Method Engineering for our ISEE (Grundy et al., 1996c). 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.

Serendipity has also been used to support process modelling for a suite of office automation programs, including Macintosh versions of Microsoft Word™, Microsoft Excel™, the GlobalFax™ fax/OCR application, and the Eudora™ email utility.

7. Architecture and Implementation

Serendipity and the tools that we have integrated to make up the ISEE described in Section 6 are implemented using the MViews architecture for developing ISDEs. Integration of Serendipity and these tools, although all independently developed, has proved very successful, due to the component and event-based nature of MViews. The integration between Serendipity and the Office Automation tools was more limited, due to the more limited interface provided by these tools.

7.1. MIEWS

MViews is a framework of object-oriented classes which provides abstractions for implementing ISDEs (Grundy and Hosking, 1996; Grundy and Hosking, 1993; Grundy et al., 1996d). New environments are constructed by specialising classes to describe the ISDE repository and view representations. 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 view" data and event translators. Figure 19 shows an example of the MViewsER Entity-Relationship and relational schema modelling tool developed using MViews (Grundy et al., 1995b). The repository describes entities, relationships and attributes, entity-relationship connections, and schema text. The multiple views provided by MViewsER include graphical ER views and textual schema and documentation views.

MViews uses an event-based software architecture. This supports 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. It is propagated to all relationships the component participates in. Receiving relationships can respond to a change description by applying operations to themselves or other components, forwarding the change description to related components, or ignoring the change. 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 et al., 1996d).

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 et al., 1996d). Snart is a persistent language, with repository and view objects dynamically saved and loaded to a persistent object store.

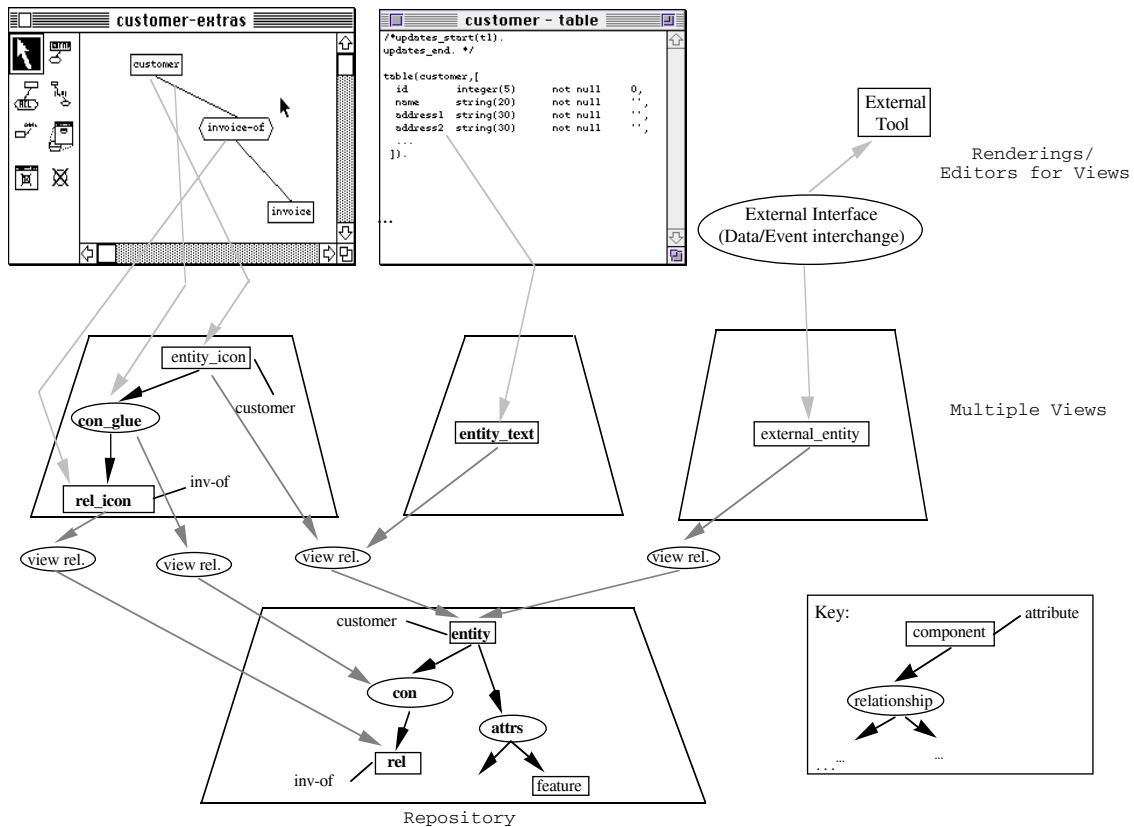


Figure 19. Example of implementing MViewsER using MViews.

7.2. INDIVIDUAL MODELLING TOOLS

We have built many ISDEs using MViews including SPE (Smart Programming Environment), MViewsER, MViewsNIAM and MViewsDP, the tools integrated into the ISEE. Integration of these tools was achieved using integrated, hierarchical repositories (Grundy and Venable, 1995a; Grundy and Venable, 1995b). This involves either defining integrated repositories which represent, for example, OOA, NIAM and ER data in a common representation, or linking individual repository items with MViews inter-repository relationships, such as between MViewsER schema fields and MViewsDP form components. 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 was built 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.

When a process model is used, process model component state variables are modified to indicate receipt of 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 20 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, and broadcasting of this modified change description is detected by the Serendipity environment base view (3), which then 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. Any change to the enactment status of Serendipity base process model components (17) is detected by their view components (18), which are re-rendered to reflect the change.

External tools, such as Microsoft 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).

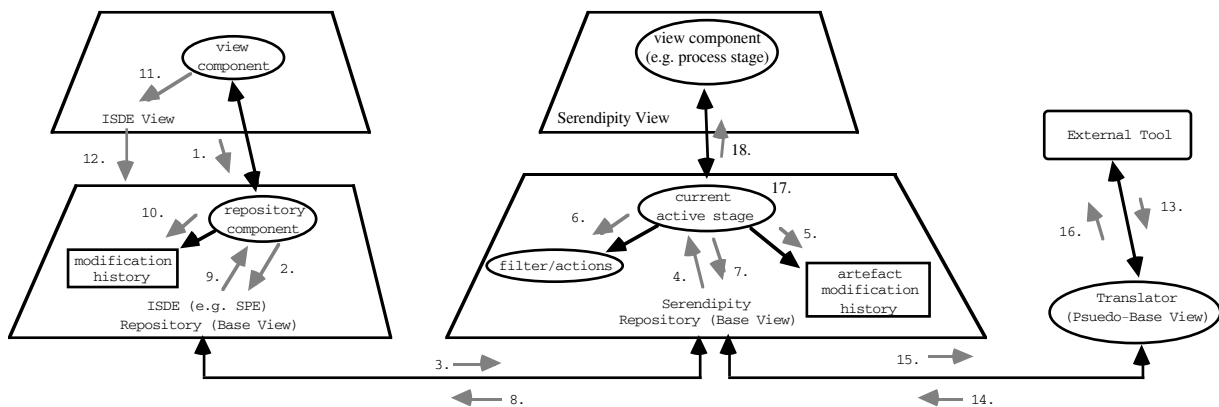


Figure 20. Integrating Serendipity with integrated tool views.

Serendipity's collaborative editing facilities are provided by the C-MViews extensions to MViews (Grundy et al., 1995c). 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.

The CSCW messaging and note annotation tools provided by Serendipity are separately-developed tools which can be used with any MViews environment. The context-sensitive messages store information about the MViews component(s) they are related to, and notes are connected to specified components by relationships. The high-level highlighting of Serendipity process model stages enacted by other users is implemented by having the central C-MViews server broadcast all enactment events to each user's environment. This then records which stages other users have enacted or deenacted, and highlights these on visible process model views (if requested by the user). We slightly modified these tools to utilise information about the current enacted Serendipity process stage for users when storing notes, sending

messages or sending text chat lines, thus supplying work context information for inter-person communication and notes.

8. Discussion

Our work has made four main contributions to Process Technology. EVPL provides an expressive, graphical process modelling and work planning language, which can be animated to support both high-level and low-level work context awareness during cooperative work. VEPL is a novel, graphical event handling language that is both accessible for simple use by inexperienced process modellers, and yet expressive enough for experienced process modellers and environment implementers to build sophisticated event handling systems. The Serendipity environment provides multiple view support for building, enacting, reusing and improving EVPL and VEPL process models, and includes a range of cooperative work capabilities. Our integration of Serendipity with other tools, particularly software development and CSCW tools, provides novel user interface capabilities, with Serendipity allowing EVPL and VEPL to utilise events from these integrated tools in various ways.

8.1. EXPERIENCE WITH SERENDIPITY

Serendipity has been applied to a range of small to medium process modelling, work planning and coordination, and tool integration problems. The authors have built a number of process models describing: software processes; coordination of collaborative software development; coordination of use of disparate office automation applications; general work processes for academics, students and business people; and for meta-models for method engineering and process improvement. Serendipity has also been used by colleagues and students to describe: software processes; steps in different Information Systems methodologies; and general work processes. The largest process model so far developed has over 200 process stages, artefacts, tools and roles, with over 60 process model and filter/action views. A project is currently in progress to more formally compare Serendipity's languages to other workflow and process modelling languages and to evaluate the Serendipity environment facilities against those of other workflow management systems and process-centred environments.

Serendipity environment performance over the above range of tasks has generally been good, but with some deficiencies, elaborated below. The asynchronous and semi-synchronous CSCW tools and editing capabilities are quite usable in our current implementation, but the synchronous tools have prohibitive performance problems (Grundy et al., 1995c). Processing stages with several filters and actions also cause performance problems, particularly with hierarchical filters (Σ annotation on event flows), which must receive events from all subprocess components. Users can request that filters and actions be actioned semi-synchronously and that the environment process filters for specified stages in idle time. These techniques improve performance with the drawback that enforcement strategies are applied some time after enactments or artefact changes have been made, necessitating occasional roll-back of artefact changes.

Integration of Serendipity with other, independently developed, MViews tools has been successful, producing tightly-integrated environments. Serendipity actions can send events to other MViews tools to perform almost any operation, and any tool event can be detected and acted upon by Serendipity filters. This is due to the component and event-based architecture of MViews systems, and the common implementation language and view and repository representation techniques. Integration with heterogeneous tools not implemented with MViews has resulted in more limited forms of integration. Serendipity actions can be defined to communicate with these tools but for significant levels of integration, this requires extensive coding and still results in less than complete integration. Serendipity currently runs on the Macintosh only. While we have integrated it in a limited way with a number of commonly-available Macintosh applications, it is much more difficult to communicate with applications on other platforms.

8.2. COMPARISON TO OTHER LANGUAGES AND SYSTEMS

Process modelling languages need to support the description of stages in the process, the data used by process model stages, and the handling of events related to stages, work artefacts and tools. Many process modelling languages use only textual languages to codify tasks and their data. Examples include those of Marvel (Heineman et al., 1992), Adele (Belkhatir et al., 1994), ConversationBuilder (Kaplan et al., 1992a; Kaplan et al., 1992b), Oz (Ben-Shaul and Kaiser, 1994a) and EPOS (Jaccheri and Conradi, 1993; Conradi

et al., 1994). Work artefacts are usually coded as types, for example in EPOS these are “dataentity” specialisations and in Adele they are aggregate type relations. Steps in a process model are usually encoded as rules, for example in EPOS as task types and Adele as event triggers. While these languages allow users to precisely specify process model data and activities, they are difficult for many users to understand and modify (Baldi et al., 1994; Bogia and Kaplan, 1995; Swenson et al., 1994), and cannot readily be utilised for high-level work context awareness, such as via animation. Our EVPL and VEPL process modelling languages have the same expressive power as these textual languages, but use a primarily graphical approach to visualising process model activities and data (with detailed process stage, artefact, tool and role characteristics specified in forms and textual views). EVPL diagrams provide significantly clearer representations of process model structure than their textual counterparts, as process modellers can see multiple levels of abstraction using EVPL diagrams. VEPL provides a graphical approach to expressing event handling, allowing process modellers to better visualise event flow and handling than a comparative textual encoding.

A variety of graphical process modelling languages have been developed for use by process-centred environments and workflow management systems. Examples include E³ p-draw’s E³ PML (Baldi et al., 1994), SPADE’s SLANG (Bandinelli et al., 1994; Bandinelli et al., 1993; Bandinelli et al., 1996), ProcessWEAVER’s transition nets (Fernström, 1993), Action Workflow’s loops (Medina-Mora et al., 1992), Regatta and TeamFLOW’s Visual Planning Language (VPL) (Swenson, 1993), and wORlds’s obligation nets (Bogia and Kaplan, 1995). Graphical modelling capabilities of some of these systems, such as VPL, SLANG and wORlds, is limited to specification of the stages in a process and the “enactment flow” between these stages. For example, SLANG encodes activity structure as a form of petri-net, but textually encodes activity artefacts and tools as process types and transitions as guards and actions; VPL represents process stages graphically with interconnecting flows, but lacks representations of stage data and enactment rules; and wORlds represents aspects of social structure graphically, including obligations between “process stages”, but provides only a simplistic graphical representation of process data and obligation rules. Our work, and work with E³ PML, ProcessWEAVER and Action Workflow, has found graphical modelling of process stage data (artefacts, tools and roles), and the visualisation data shared by multiple process stages, to be very important when using complex process models. E³ PML uses an object-oriented process modelling notation, with a variety of representations (stages, artefacts, tools), and interconnections (task decomposition, inheritance, control flow). E³ PML represents task decomposition on the same diagram, whereas EVPL utilises multiple overlapping and hierarchical views, making decomposition easier to manage and resulting in clearer process models. E³ PML also lacks the range of usage connections and annotations of EVPL, making it less expressive. ProcessWEAVER (Fernström, 1993) uses a transition net (a form of petri-net) to model cooperative procedures via tasks and subtasks, with token flow driving enacted process advancement. The transition nets do not explicitly represent artefacts, tools and roles, but nodes can be defined which correspond to the manipulation of them. ProcessWEAVER nets thus do not make clear the roles involved in stages of a process, nor the artefacts and tools, nor do they indicate usage of this work context information, as in EVPL. Action Workflow (Medina-Mora et al., 1992) uses a document flow model to represent work processes, which has been found to be deficient for many process modelling situations (Swenson et al., 1994). EVPL uses a more flexible and expressive state-transition model, as do VPL, ProcessWEAVER and SPADE, with EVPL processes driven by the propagation of enactment events. This notion of explicit enactment events in EVPL, rather than state transition by token propagation in SPADE and ProcessWEAVER or the use of speech acts in VPL, is important, as it is utilised by VEPL to handle enactment, artefact update and tool events in a homogeneous fashion.

The event-handling of most graphical process modelling languages is codified graphically for “enactment” events (state transitions), but textually for stage guards and actions, and for coding interaction with people or other tools. Most workflow management systems, such as Action Workflow, Regatta and TeamFLOW, provide a limited range of “interesting events”, supporting interaction with other tools and some forms of notification and work coordination based on event occurrence. These are usually codified using a form-based approach where modellers specify simple actions to carry out based on a range of possible events. VEPL provides a more powerful, clearer and extensible event-handling language than this approach. VEPL filter/actions can handle more complex events, provide a wider range of actions, be extended by building hierarchical models (or using an API), and have graphical representations which more clearly show event flow in a process model. Most process modelling languages use textual rules with guard predicates and executable actions to specify how events are handled. ProcessWEAVER provides a textual co-shell language which allows users to specify actions for process model nodes when fired by input tokens. SLANG uses textual specifications of guards and actions for nodes in a state transition network. Marvel

and Oz use guarded rules specified over data types. Adele provides an activity manager, which uses a textual language to specify database-related event handling for process models. These approaches all use textual languages, and hence complex rules suffer from the same problems as when presenting other process model structures textually. The relationships between different stages, tools and artefacts are expressed in a linear, textual fashion, unlike graphical VEPL event handling models where event flows and process model artefacts are more clearly represented in graph form. This is a particular problem for rules which span multiple process stages, as understanding rule behaviour requires several textual data and task specifications to be viewed. VEPL permits specification of both simple and very complex event handling graphically, resulting in high-level visualisations of event handling behaviour. For example, the specification of a simple work coordination mechanism where a user is notified whenever another user modifies a particular work artefact, shown in Figure 11, results in an intuitive event handling model which is relatively easy for users to understand and modify. By allowing previously-defined filter/action templates to be composed using the same event propagation notation, VEPL supports reuse of a wide range of event handling behaviour. In contrast, approaches which use textual codification to handle these kinds of events, produce event handlers that are less clear and generally less reusable than those of VEPL. Visual dataflow-based languages, such as Fabrik (Ingalls et al., 1988) and Prograph (Cox et al., 1989), provide graphical dataflow models which are similar in nature to VEPL, but use dataflow rather than event-flow, which is less appropriate in a process modelling domain. Some visual languages, such as ViTABaL (Grundy and Hosking, 1995), utilise an event-driven model but lack the equivalent of Serendipity's filters, actions, and interest specification capabilities. Because of their general-purpose nature, these visual programming languages lack specific process modelling capabilities, and thus can not express and represent process model event-handling and work coordination tasks as effectively as EVPL and VEPL.

Most process modelling languages are supported by their own PCE or Workflow Management System (WFMS). For example E³ PML is supported by the E³ p-draw tool (Jaccheri and Gai, 1992), SLANG is supported by the SPADE PCE (Bandinelli et al., 1994; Bandinelli et al., 1993; Bandinelli et al., 1996), VPL by the Regatta (Swenson et al., 1994) and TeamFLOW (TeamWARE, 1996) WFMSs, and obligation nets by the wOrlds CSCW environment (Bogia and Kaplan, 1995). These environments typically allow process models to be enacted and advanced as users complete work on tasks and subtasks. Environments utilising textual languages are limited in how they can inform users of the state of enacted process models. Most avoid using the textual codification of the process model to visualise their state. In contrast, EVPL process models are animated to give users feedback on process model enactment and other users' work. Action Workflow, wOrlds, Regatta and TeamFlow also use highlighting of enacted process models. However in Serendipity, we have gone further than these systems, and highlight stages as they are enacted/deenacted, highlight stages of cooperating users, and highlight artefacts, tools and roles currently being used by an enacted process stage (Grundy et al., 1996b).

Most recent Computer Supported Cooperative Work (CSCW) research has focused on low-level interaction mechanisms, such as synchronous and asynchronous editing. Examples include most Groupware systems (Ellis et al., 1991), GroupKit (Roseman and Greenberg, 1996), Mjølner (Magnusson et al., 1993), C-MViews (Grundy et al., 1995c), and Rendezvous (Hill et al., 1994). These systems lack information about the "work context" changes have been carried out in, whereas Serendipity makes this information readily accessible to users. There is some work on providing higher-level process modelling and coordination facilities, such as workflow configuration (Medina-Mora et al., 1992), obligations (Kaplan et al., 1992b, Bogia and Kaplan, 1995), and shared workspace awareness (Roseman and Greenberg, 1996), but these systems are generally separate from the work artefacts or editing tools, and are not used to provide work context information. In Serendipity the text chats and note annotation CSCW tools incorporate information from the enacted process models, as do other tools integrated with Serendipity. This provides users with more context awareness capabilities than are supported by other CSCW and process modelling systems.

Oz (Ben-Shaul and Kaiser, 1994a) enables the definition of high-level work coordination capabilities, such as summits and treaties, utilising extensions to the Marvel rule-based model (Heineman et al., 1992), and supports cooperative transactions for itself and integrated tools (such as ProcessWEAVER) via an external concurrency control architecture (Heineman and Kaiser, 1995). TeamFLOW, wOrlds and ConversationBuilder use obligations between process stages to enable enactment events to be propagated between process stages from different diagrams. ProcessWEAVER (Fernström, 1993) uses transition networks to model synchronisation of concurrent activities by cooperative agents, but like wOrlds these deal mainly with "enactment"-style events. VEPL supports the definition of complex work coordination by using EVPL constructs in its event-handling models. We have used VEPL to build a variety of low-level concurrency control mechanisms, typically supported by groupware tools. We have also used it to build higher-level work coordination models similar to those used by ProcessWEAVER and wOrlds (Grundy et

al., 1996b). Serendipity uses both a shared server to broadcast events and share data, and a repository storing local process model and work artefacts. This allows it to support Oz-style summits, with process model enactment events generated by multiple users broadcast via the shared server. VEPL models can handle and constrain data sharing, in a similar manner to the “diplomats” utilised by Oz treaties, and can also support concurrent transactions by having actions store sequences of artefact changes and undo them if multiple transaction conflicts arise.

There have been several attempts at integrating CSCW, ISDEs and workflow/PCEs, including ConversationBuilder (Kaplan et al., 1992b), MultiviewMerlin (Marlin et al., 1993), wOrlds (Bogia and Kaplan, 1995), SPADE/ImagineDesk (Di Nitto and Fuggetta, 1995; Bandinelli et al., 1996), ProcessWEAVER and Oz (Heineman and Kaiser, 1995), and various groupware tools and Oz (Ben-Shaul et al., 1994b; Ben-Shaul and Kaiser, 1996). Many of these, such as MultiviewMerlin, SPADE/ImagineDesk and wOrlds, have produced environments with a high degree of integration. However none have produced environments which capture detailed information about enacted process models and present this “work context” information using the integrated tools themselves, nor do they allow information from the integrated tools to be utilised in the process modelling environment exactly as though the tools were part of that environment. In contrast in our integrated environment the boundaries of the two systems, from a user's perspective, have disappeared, despite there being no modifications to the underlying architectures of either system. The augmentation of SPE and CSCW tool data with Serendipity process model information, and the highlighting of in-use SPE artefacts by Serendipity, allows users to remain aware of when and why other users have made artefact changes or caused tool events to occur. Similarly, SPE artefacts (classes, diagrams etc.) and tools (OOA/D editor, text editor, debugger etc.) can be represented in Serendipity as EVPL artefacts and tools, and VEPL filter/actions can use these artefacts and tools as both producers of events and as event filters. Serendipity process stages can also store work histories using the event descriptions generated by the integrated tools. This provides closer user interface integration between process modelling tool, CSCW tools and work tools than do these other PCE, CSCW environment and work tool integration efforts.

Our integration of Serendipity, MViews CSCW tools, and an ISEE has been very successful, due to the event-driven and component-based natures of these environments and their common implementation platform. We have achieved a much lesser degree of integration of Serendipity with third-party tools, because of limitations in the interfaces provided to the tool data and control functions. Recent work on integrating such heterogeneous software development, CSCW and process modelling environments has focused on data aspects (e.g. federated approaches to tool integration (Bounab and Godart, 1995)), control integration via enveloping (e.g. between Oz and third party tools (Valetto and Kaiser, 1995)), and process integration (e.g. coordinating tool usage via PCEs (Marlin et al., 1993; Di Nitto and Fuggetta, 1995; Bandinelli et al., 1996)). Serendipity provides, via MViews and VEPL, an event-driven interface which supports control and process integration with disparate tools. At present, Serendipity utilises a repository which stores process model data in the form of Prolog terms. While Serendipity actions can be defined to facilitate import/export of this data with heterogeneous tools, this does involve more work than utilised in federated approaches to tool integration via distributed data management (Bounab and Godart, 1995; Valetto and Kaiser, 1995).

8.3. CURRENT AND FUTURE WORK WITH SERENDIPITY

The experience of the authors and other users of Serendipity to date has been generally positive, but a number of desirable improvements to the languages and environment have been identified. The EVPL and VEPL languages have proved concise and yet expressive for a large range of process modelling and work coordination/event-handling problems. There is a need, however, to support better artefact structuring and repository querying facilities for users. To this end we are currently developing a third visual notation for Serendipity, the Visual Query Language (VQL). This will allow users to specify the structure of artefacts using an MViews-style component and relationship model. Users will also be able to define complex, graphical queries over MViews (and other tool) environment repositories. Triggering of filters and actions will then be possible over the query result, which will be incrementally updated as data is modified.

We have been working on filters and actions which allow work contexts to be determined dynamically, in addition to being specified using artefact, role and tool representations. This will allow Serendipity to determine changes of context (i.e. change of current enacted process stage) automatically, rather than requiring users to do this manually, as at present. Actions will utilise the history of work context information for stages to determine such context changes. Visualisation techniques allowing users to view

summaries of inter-related changes made in different contexts are also being developed. These facilities will, we believe, assist Serendipity in better supporting more informal aspects of work, which is currently poorly supported by existing workflow and PCEs (Kaplan et al., 1996).

We are currently porting Serendipity to Java and are utilising a Web-based interface to the modelling language views (Grundy et al., 1997). This will improve the portability of our environment. More significantly, this will enable us to provide better integration with heterogeneous tools, or at least utilise the now commonly-understood “plug-in” and “helper application” model of loose integration employed by browser-style applications. Another advantage will be the ability to reuse other people’s CSCW and work tools, in the form of Java applets, plug-ins or helper applications. We have found that systems built on event-based architectures, such as those of MViews, are much more amenable to tool integration, and plan to use Serendipity’s event-handling language in conjunction with such systems to better facilitate their integration with Serendipity.

9. Summary

Our work with Serendipity makes a number of new contributions to research into process technology. EVPL, while having some similarities to other graphical process modelling languages, utilises a versatile enactment event-based execution model. EVPL is used for generic process modelling, meta-process modelling and detailed work planning, and supports process stage work context modelling via representations of artefacts, tools, role communication and various kinds of inter-component usage relationships. VEPL is a novel event-handling notation which utilises EVPL model components, as well as introducing filters and actions, to handle enactment, artefact update, tool-induced and role-induced events. EVPL and VEPL together allow process modellers to specify arbitrary event-handling for process models, which includes specifying a large variety of work coordination strategies, automatic process model rule application, and propagation and storage of events between stages, artefacts, tools and roles.

Serendipity provides a support environment for EVPL and VEPL which includes multiple views of process models and event handling specifications. Enactment of process models utilises highlighting of enacted stages, event flows and artefact, tool and role representations, helping multiple users to keep aware of others’ work contexts. Collaborative editing of process models is supported, along with CSCW tools for annotating, messaging and talking about models. Integration of Serendipity and MViews-based tools for performing work results in environments with highly integrated user interfaces. Descriptions of changes made in other tools are annotated with work context (i.e. process stage) information, process stages store lists of changes made while they are enacted (forming histories of work), and icons in both Serendipity and the integrated tools are highlighted to facilitate group awareness. Such tight user interface integration is not supported by most other PCE/workflow and work tool integration efforts. Serendipity utilises a component, event-based architecture. A hybrid technique is used to store process model information in local repositories for speed of access, and a central repository used to enable sharing of versions and broadcasting of enactment, artefact and tool events.

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. We are currently porting MViews to Java and will also port Serendipity and the MViews suite of software development tools, to improve their accessibility, performance, and ability to integrate existing tools. Filters and actions are currently interpreted, but will be compiled to Java to improve their performance.

Acknowledgments

The authors gratefully acknowledge the many helpful comments of the anonymous reviewers on earlier drafts of this paper.

References

- Baldi, M., Gai, S., Jaccheri, M.L., and Lago, P. 1994. Object Oriented Software Process Design in E³. In *Software Process Modelling & Technology*, eds. A. Finkelstein and J. Kramer and B. Nuseibeh, Research Studies Press, 1994.
- Bandinelli, S., Fuggetta, A., and Ghezzi, C. 1993. Process model evolution in the SPADE environment. *IEEE Transactions on Software Engineering*, vol. 19, no. 12, pp. 1128-1144.

- Bandinelli, S., Fuggetta, A., Ghezzi, C., and Lavazza, L. 1994. SPADE: an environment for software process analysis, design and enactment. In *Software Process Modelling & Technology*, eds. A. Finkelstein and J. Kramer and B. Nuseibeh, Research Studies Press, 1994.
- Bandinelli, S., Di Nitto, E., and Fuggetta, A. 1996. Supporting cooperation in the SPADE-1 environment, *IEEE Transactions on Software Engineering*, vol. 22, no. 12.
- 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, 1992, pp. 21-31.
- Belkhatir, N., Estublier, J., and Melo, W.L. 1994. The Adele/Tempo Experience. In *Software Process Modelling & Technology*, eds. A. Finkelstein and J. Kramer and B. Nuseibeh, Research Studies Press, 1994.
- Ben-Shaul, I.Z. and Kaiser, G.E. 1994a. A Paradigm for Decentralized Process Modeling and its Realization in the Oz Environment. In *Sixteenth International Conference on Software Engineering*, IEEE CS Press, pp. 179-188.
- Ben-Shaul, I.Z., Heineman, G.T., Popovich, S.S., Skopp, P.D. and Tong, A.Z., and Valetto, G. 1994b. Integrating Groupware and Process Technologies in the Oz Environment. In *9th International Software Process Workshop: The Role of Humans in the Process*, Ghezzi, C., IEEE CS Press, Airlie, VA, October 1994, pp. 114-116.
- Ben-Shaul, I.Z. and Kaiser, G.E. 1996. Integrating Groupware Activities into Workflow Management Systems. In *7th Israeli Conference on Computer Based Systems and Software Engineering*, Tel Aviv, Israel, June 1996, pp. 140-149.
- Bogia, D.P. and Kaplan, S.M., Flexibility and Control for Dynamic Workflows in the wOrlds Environment. In *Proceedings of the Conference on Organisational Computing Systems*, ACM Press, Milpitas, CA, November 1995.
- Bounab, M. and Godart, C. 1995. A Federated Approach to Tool Integration. In *Proceedings of CAiSE'95*, Springer-Verlag, LNCS 932, Finland, June 13-16 1995, pp. 269-282.
- Conradi, R., Hagaseth, M., Larsen, J., Nguyen, M.N., Munch, B.P., Westby, P.H., Zhu, W., Liu, C. 1994. *EPOS: Object Oriented Cooperative Process Modeling*. In *Software Process Modelling & Technology*, eds. A. Finkelstein and J. Kramer and B. Nuseibeh, Research Studies Press, 1994.
- Cox, P.T., Giles, F.R., and Pietrzykowski, T. 1989. Prograph: a step towards liberating programming from textual conditioning, , IEEE Computer Society Press. In *Proceedings of the 1989 IEEE Workshop on Visual Languages*, 1989, pp. 150-156.
- Di Nitto, E. and Fuggetta, A. 1995. Integrating process technology and CSCW. In *Proceedings of IV European Workshop on Software Process Technology*, LNCS, Springer-Verlag, Leiden, The Netherlands, April 1995.
- Ellis, C.A., Gibbs, S.J., and Rein, G.L. 1991. Groupware: Some Issues and Experiences, *Communications of the ACM*, vol. 34, no. 1, 38-58, January 1991.
- Fernström, C. 1993. ProcessWEAVER: Adding process support to UNIX. In *2nd International Conference on the Software Process: Continuous Software Process Improvement*, IEEE CS Press, Berlin, Germany, February 1993, pp. 12-26.
- Grundy, J.C. and Hosking, J.G. 1993. A framework for building visual programming environments. In *Proceedings of the 1993 IEEE Symposium on Visual Languages*, IEEE Computer Society Press, 1993, pp. 220-224.
- Grundy, J.C., Mugridge, W.B., Hosking, J.G., and Apperley, M.D. 1995a. 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., Fenwick, S., and Mugridge, W.B. 1995b. Connecting the pieces, In *Visual Object-Oriented Programming*, eds. M. Burnett, A. Goldberg, T. Lewis, Manning/Prentice-Hall, 1995.
- the 6th European Workshop on Next Generation of CASE Tools*, Finland, June 1995, pp. 109-116.
- Grundy, J.C., Mugridge, W.B., Hosking, J.G., and Amor, R. 1995c. 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. and Hosking, J.G. 1995. ViTABaL: A Visual Language Supporting Design By Tool Abstraction. In *Proceedings of the 1995 IEEE Symposium on Visual Languages*, IEEE CS Press, Darmstadt, Germany, September 1995, pp. 53-60.
- Grundy, J.C. and Venable, J.R. 1995a. Providing Integrated Support for Multiple Development Notations. In *Proceedings of CAiSE'95*, Springer-Verlag, LNCS 932, Finland, June 1995, pp. 255-268.
- Grundy, J.C., and Venable, J.R. 1995b. Developing CASE tools that support integrated design notations. In *Proceedings of Grundy, J.C., Hosking, J.G., and Mugridge, W.B. 1996a. Towards a Unified Event-based Software Architecture*. In *Joint Proceedings of the SIGSOFT'96 Workshops*, eds. L. Vidal, A. Finkelstein, G. Spanoudakis, and A.L. Wolf, ACM Press, October 14-15 1996, pp. 121-125.
- Grundy, J.C., Hosking, J.G., and Mugridge, W.B. 1996b. Low-level and high-level CSCW in the Serendipity process modelling environment. In *Proceedings of OZCHI'96*, IEEE CS Press, Hamilton, New Zealand, Nov 24-27 1996.
- Grundy, J.C., Venable, J.R., Hosking, J.G., and Mugridge, W.B. 1996c. Coordinating collaborative work in an integrated Information Systems engineering environment. In *Proceedings of the 7th Workshop on the Next Generation of CASE tools*, Crete, 20-21 May 1996.
- Grundy, J.C., Hosking, J.G., and Mugridge, W.B. 1996d) Supporting flexible consistency management via discrete change description propagation, *Software - Practice & Experience*, vol. 26, no. 9, 1053-1083, September 1996.
- Grundy, J.C., Mugridge, W.B., and Hosking, J.G. 1996e. A Java-based toolkit for the construction of multi-view editing systems. In *Proceedings of the Second Component Users Conference*, Munich, July 14-18 1997.
- Grundy, J.C. and Hosking, J.G. 1996. Constructing Integrated Software Development Environments with MViews, *International Journal of Applied Software Technology*, vol. 2, no. 3-4, pp. 133-160.
- Grundy, J.C., Hosking, J.G., and Mugridge, W.B. 1997. A Visual, Java-based Componentware Environment for Constructing Multi-view Editing Systems, In *Proceedings of 2nd Component Users Conference*, Munich, July 1997.
- 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*, IEEE CS Press, Brisbane, December 1995, pp. 430-438.

- Heineman, G.T., Kaiser, G.E., Barghouti, N.S., and Ben-Shaul, I.Z. 1992. Rule Chaining in Marvel: Dynamic Binding of Parameters, *IEEE Expert*, vol. 7, no. 6, 26-32, December 1992.
- Heineman, G.T. and Kaiser, G.E. 1995. An Architecture for Integrating Concurrency Control into Environment Frameworks. In *Proceedings of the 17th International Conference on Software Engineering*, IEEE CS Press, Seattle, Washington, April 1995, pp. 305-313.
- Hill, R.D., Brinck, T., Rohall, S.L., Patterson, J.F., and Wilner, W. 1994. The Rendezvous Architecture and Language for Constructing Multi-User Applications, *ACM Transactions on Computer-Human Interaction*, vol. 1, no. 2, June 1994, 85-125.
- Ingalls, D., Wallace, S., Chow, Y.Y., Ludolph, F., and Doyle, K. 1988. Fabrik: A Visual Programming Environment. In *Proceedings of OOPSLA '88*, ACM Press, pp. 176-189.
- Jaccheri, M.L. and Gai, S. 1992. Initial Requirements for E3: an Environment for Experimenting and Evolving Software Processes. In *Proceedings of EWSPT'92*, Trondheim, Norway, September 1992, pp. 99-102.
- Jaccheri, M.L. and Conradi, R. 1993. Techniques for Process Model Evolution in EPOS, *IEEE Transaction on Software Engineering: Special issue on Software Process Evolution*, vol. 19, no. 12, 1145--1156, December 1993.
- Kaplan, S.M., Tolone, W.J., Carroll, A.M., Bogia, D.P., and Bignoli, C. 1992a. Supporting Collaborative Software Development with ConversationBuilder. In *Proceedings 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. 1992b. Flexible, Active Support for Collaborative Work with ConversationBuilder. In *1992 ACM Conference on Computer-Supported Cooperative Work*, ACM Press, pp. 378-385.
- Kaplan, S.M., Fitzpatrick, G., Mansfield, T., and Tolone, W.J. 1996. Shooting into Orbit. In *Proceedings of Oz-CSCW'96*, DSTC Technical Workshop Series, University of Queensland, Brisbane, Australia, August 1996, pp. 38-48.
- 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. In *Proceedings of the 6th International Software Process Workshop*, IEEE CS Press, Hokkaido, Japan, 28-31 October 1990.
- Krishnamurthy, B. and Hill, M. 1994. CSCW'94 Workshop to Explore Relationships between Research in Computer Supported Cooperative Work & Software Process. In *Proceedings of CSCW'94*, ACM Press, pp. 34-35.
- Magnusson, B., Asklund, U., and Minör, S. 1993. Fine-grained Revision Control for Collaborative Software Development. In *Proceedings of the 1993 ACM SIGSOFT Conference on Foundations of Software Engineering*, Los Angeles CA, December 1993, pp. 7-10.
- Marlin, C., Peuschel, B., McCarthy, M., and Harvey, J. 1993. MultiView-Merlin: An Experiment in Tool Integration. In *Proceedings of the 6th Conference on Software Engineering Environments*, IEEE CS Press.
- Medina-Mora, R., Winograd, T., Flores, R., and Flores, F. 1992. The Action Workflow Approach to Workflow Management Technology. In *Proceedings of CSCW'92*, ACM Press, pp. 281-288.
- Roseman, M. and Greenberg, S. 1996. Building Real Time Groupware with GroupKit, A Groupware Toolkit, *ACM Transactions on Computer-Human Interaction*, Vol 3, No. 1, March 1996, 1-37.
- Swenson, K.D. 1993. A Visual Language to Describe Collaborative Work. In *Proceedings of the 1993 IEEE Symposium on Visual Languages*, IEEE CS Press, 1993, 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*, vol. 1, no. 1.
- TeamWARE, Inc. 1996. *TeamWARE Flow*, <http://www.teamware.us.com/products/flow/>.
- Tolone, W.J., Kaplan, S.M., and Fitzpatrick, G. 1995. Specifying DynamicSupport for Collaborative Work within wOrlds. In *Proceedings of the 1995 ACM Conference on Organizational Computing Systems*, Milpitas, CA, August 1995, pp. 55-65.
- Valetto, G. and Kaiser, G.E. 1995. Enveloping Sophisticated Tools into Computer-Aided Software Engineering Environments. In *IEEE Seventh International Workshop on Computer-Aided Software Engineering*, July 1995, pp. 40-48.
- 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 Conference*, Springer-Verlag, LNCS 1021, Gold Coast, Australia, 1995.