Leveraging Usage History to Support Enterprise System Users

Tamara Babaian and Wendy Lucas
Bentley University, Waltham, MA 02452, U.S.A.

Keywords: Human-computer Collaboration, Enterprise Systems, ERP, Human-computer Interaction, Usability.

Abstract: Users of Enterprise Information Systems face considerable challenges in relating the generic, all-encompassing system interfaces to the vocabulary and business practices of their organizations. The help functionality is often too general to be of much use, and employees typically prefer to ask a colleague or a help-desk consultant for directions on how to proceed rather than turn to a user manual. A more direct, less resource-intensive approach than person-to-person assistance is to have the system itself provide the guidance the user is seeking in how to navigate and use its interface. We have implemented a playback interface that aids users of our Enterprise Resource Planning prototype in learning to perform an individual task or a multi-task business process. Our algorithm creates dynamic visualizations of previously occurring system-user interactions for demonstrating the system interface. The demonstrations are constructed in real time based on usage log data aggregated from multiple, pertinent user sessions. We discuss the challenges of identifying and aggregating relevant usage log records for task demonstrations and highlight the components of our data model designed to overcome these challenges. The playback interface offers a natural, low-cost alternative that is more flexible than pre-recorded tutorials, as the user can select a tutorial from a variety of options. In addition, it has the advantage of representing the actual business practices within the organization, which may differ from those prescribed by the system.

1 INTRODUCTION AND MOTIVATION

Industry reports (Hestermann, 2009; Hamerman, 2007; Iansiti, 2007) and field studies (Topi et al., 2005; Cooprider et al., 2010) of Enterprise Resource Planning (ERP) system usage have found that the users' ability to perform the appropriate actions is often hindered by the usability issues plaguing these systems. The complexity of the task interfaces, the opacity of the relationship between those interfaces and the process being performed, and the lack of navigational cues work against the effective use of these systems. Companies address this lack of usability by providing weeks or even months of training on system usage and developing an internal support structure of "super users" (in-house experts) and colleagues to whom users can turn for help. Even experienced users seek help from others when undertaking tasks they perform infrequently or accessing unfamiliar interfaces.

In this paper, we present an approach for creating on-request, dynamic visualizations of system-user interactions for supporting ERP users in learning (or recalling) how to perform a multi-task business process. The system culls data from past interactions that have been captured to a usage log and couples it with its own knowledge of its task interfaces in order to create step-by-step playbacks to the user of how she or others have previously performed a particular task or a series of tasks comprising a process.

The major challenge behind composing system demonstrations from vast amounts of usage data comes from the need to identify only those user sessions that are relevant to a process and to aggregate the key-press level data from those sessions (Ivory and Hearst, 2001) in *real-time*. The sessions typically involve multiple users who have worked on parts of the process during non-continuous time periods that may be overlapping. Since the logging facilities of existing ERP systems are not sufficient for this task, we have designed a representation of the tasks, their composition into user interface components, and their relationship to domain objects, which we call the Task-Interface-Log (TIL) model. This model and the algorithms we have developed for enabling playback have been implemented in an ERP prototype that is used here for demonstrating our approach.

Babaian T. and Lucas W..
Leveraging Usage History to Support Enterprise System Users.
DOI: 10.5220/0004001100350044
In *Proceedings of the 14th International Conference on Enterprise Information Systems* (ICEIS-2012), pages 35-44
ISBN: 978-989-8565-12-9

Enabling automated demonstrations of system usage supports the users' desire to learn from others in the organization about how the system is used in practice, which can vary significantly from the prescribed processes (Rozinat and Van der Aalst, 2008). Other benefits include:

- **Time and Cost Savings -** users can get the help they need without leaving their desks. The composition of demonstrations is done automatically by streamlining the previously captured record of a user performing a task, without the need for any help desk personnel.
- **Flexibility -** users can be given choices concerning the interactions they would like to view, including the task or process, the relevant organizational unit, a date range of interest, etc. This flexibility supports organizational memory and maintains the currency of the tutorials, as they are generated in real time from data that includes the most recent instances of business processes performed within the system.
- **In-context Help -** unlike the help functionality currently provided by ERP systems, which is typically too generic to be useful, our demonstrations capture the context of the system-user interaction and are accessible directly from the system interface.

The next section of this paper discusses related work. In section 3, we highlight components of the TIL model that are essential to task and process instance identification. The playback interface is described in section 4. Conclusions and directions for future research are presented in Section 5.

2 RELATED WORK

In acknowledging the need for dramatic improvements in tutorial and help methods, it is noted in (Plaisant and Shneiderman, 2005) that recorded demonstrations of interfaces are a very effective method for helping users learn procedural tasks. The tutorials they describe, however, are prerecorded videos that are accessed via external websites and are not integrated with a user interface. In our approach, automated tutorials are dynamically generated from logs of actual usage, thereby bypassing expenses associated with prerecording videos while providing up-to-date, in-context demonstrations of any process that has been performed with the system.

There are several techniques for providing in-context help with complex interfaces. The focus of many of these approaches is on the interface components themselves. ToolClips (Grossman and Fitzmaurice, 2010) helps users understand how to use a tool or function by augmenting traditional textual tooltips with video and documentation content. AIMHelp (Chakravarthi et al., 2009) provides dynamic, context-sensitive help for the AIM (Auckland Interface Manager) GUI-based application, including an automated help index, a browsable event log for listings of prior events, and displays for the causes of event triggers and the effects of a selected widget. The CogentHelp prototype tool views the help system as "one topic per widget" (Caldwell and White, 1997), with human-generated snippets attached to widgets in user interfaces built using the Java Abstract Window Toolkit.

A creation tool for providing contextual help for GUIs using screenshots is presented in (Yeh et al., 2011). Various approaches exist for guiding user interactions through a process. The SmartAidè tool (Ramachandran and Young, 2005) provides context-sensitive help to novice users of complex interfaces based on AI planning. It gives step-by-step textual instructions, automatically generates action sequences for execution within the workspace, and changes the state of the interface. The CoScripter tool (Leshed et al., 2008) uses a programming-by-demonstration approach for capturing knowledge on how to perform web-based procedures. Users record their actions as editable, executable scripts that are stored in a wiki for maintaining and sharing among users.

A model-based approach is described in (Brusilovsky and Cooper, 2002), in which a hypermedia interface provides adaptive diagnostics and interactions to maintenance technicians of complex equipment that are tailored to the competency of the user. The OWL (Organization-Wide Learning) recommender system (Linton et al., 2000) uses data logged from users of instrumented versions of software, with individualized coaching in the form of recommended learning tips that are computed on a monthly basis. In (Shen et al., 2009), workflow models are applied for assisting users of desktop applications, with an intelligent workflow assistant automatically keeping track of ToDo lists and automating part of the workflow when possible.

In the ERP domain, process mining is typically used passively for workflow analysis. Its active use in an online setting is promoted in (Van der Aalst et al., 2010), which proposes the mining of partial cases that have not yet completed for use in recommending the remaining steps to take for minimizing cost and time. The approach described in (Dorn et al., 2010) provides recommendations on

the most suitable next steps from previous processes executed by the current user along with process decisions from all users involved in the same process type. Their self-adjusting user model classifies each user and weighs recommendations based on those classifications. In (Greco et al., 2005), the focus is on aiding system administrators of large-scale applications by identifying from system logs those choices performed most recently in the past that led to a desired outcome.

While our approach is also model- and process-driven, it is differentiated by our framework, which supports the automatic derivation of processes from usage logs based on records of task inputs and outputs rather than on temporal data. This reduces noise arising from multiple users working on interconnected, highly concurrent processes and avoids many of the shortcomings identified in (Van der Aalst, 2010).

Another related research area is the generation of user interfaces (UIs) from models. In (Tran et al., 2010), a semi-automatic approach based on combining task, user, and domain models for generating both the interface design and the code is described. Initial UI models and prototypes can also be generated from system domain models for representing domain entities and transactions along with use case models of intended user tasks (Da Cruz and Faria, 2010). The TOOD (Task Object Oriented Design) method is used in generating the UI from the task model (Mahfoudhi et al., 2005).

The focus of these approaches is on automating UI design for low-cost, fast implementation. While our approach also declaratively specifies the components in the interface based on a task, domain, and interface model, the focus is on improving user support.

3 CAPTURING THE USAGE DATA

For the purpose of our investigations, we have developed a prototype that implements a selection of typical ERP tasks. We briefly outline the components of its interface to provide context for the presentation of our model and algorithms that follows.

Each task interface is implemented using one or more interface pages. The graphical user interface of each page uses groups of interactive components that are designed specifically to automatically record all interactions with the user. These interactive components are input fields, standard buttons, and menus.

While the user input components can be laid out in different fashion on different pages, all user-system interactions occur via these input controls. The composition of each page is described within the data model of the system, as discussed next.

3.1 Task-Interface-Log (TIL) Model

We deploy a Task-Interface-Log (TIL) model at the core of our prototype to enable the recording and reconstruction of the details of any interaction between the system and its users. An entity-relation diagram of the essential components of the TIL model presented in Figure 2 illustrates what follows here.

Our model captures the essential description of the tasks that the system is designed for in the **Task module**. Each task description includes the specification of the type of object produced as the result of executing a task, which is described in an attribute called *DTableOut*. This is an abbreviation for Domain Table Output, since task output is always stored in a table from the ERP domain database that we call the **Domain module**. For example, the Add Material task results in the creation of a new material record, which is stored in the Material table. Thus, the value of *DTableOut* for Add Material is Material. The Task module also includes user-configurable information regarding the tasks that are included in the business processes.

Each task is implemented as a set of ordered interface pages containing user input components, such as input fields, buttons, and menus. The description of each task page is stored in the **Interface module**. Each input component record in the InputControl table includes a *DTable* attribute, which specifies if the data entered into the component must have a corresponding record in a particular domain table (i.e. a Domain module table). For example, a text field for entering plant values will have the *DTable* attribute set to Plant. Note that a collection of *DTable* attributes of all input components associated with a task specifies the task's *domain inputs*, i.e., the types of domain objects used to produce the output of the task.

The Task and Interface modules are *static*, in that their contents do not change after the system has been configured. They provide the means for creating an interface from a declarative specification as well as for identifying the possible input-output connections between tasks. These connections play an important role in identifying which task instances are potentially related to each other and which are not.

The **Logging module** records user interactions with the system on two interconnected levels: the task level and the interface level. Logging on the task level involves keeping track of *task instances*, i.e., the users' engagement with the system on a particular task. A new task instance is created whenever a user opens the task interface. A task instance can ex-

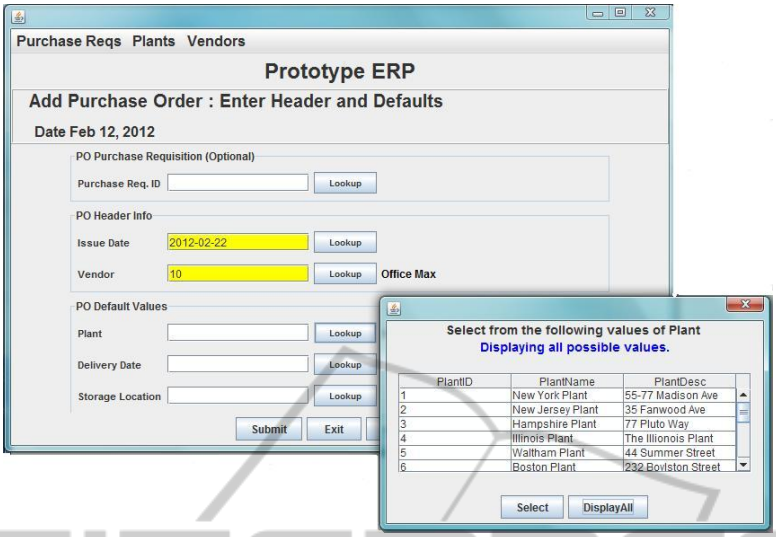

Figure 1: A screenshot demonstrating typical features of our prototype's interface.

tend over multiple user sessions and is considered unfinished until the task's output object is submitted to the domain database or the task instance is canceled. When a task instance is completed, the identifier of the created domain output object is recorded in the *OutPKVal* attribute of the TaskInstance table. Thus, each created domain object can be traced back to the particular task instance that produced (or updated) it.

The SessionTaskInstance table of the Logging module captures the chronology of each task instance within user sessions, which may have been performed by multiple users. The system is constrained to allow only one user to work on a task instance at any particular moment.

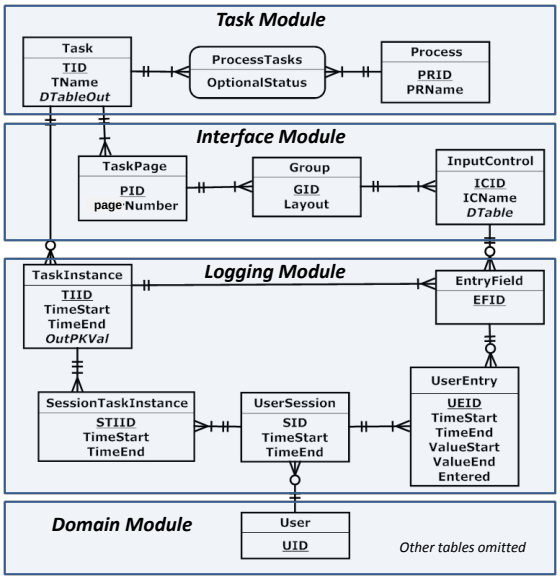

Figure 2: The Task-Interface-Log (TIL) model.

The detailed key-press level information regarding the users' interactions with input controls within the task instance is also recorded in the Logging module. Whenever a task page is rendered on the screen, the system creates a new record in the EntryField table for each input control on the page. An entry field is an instantiation of an input control within a particular user session. Each entry field is attributable to a single task instance. Users' interactions with entry fields are recorded in the UserEntry table. Every time an input control gains focus, the system records the timing of that event as well as the time of the subsequent loss of focus. It also records the content of the entry field at the time focus was gained, the input entered by the user, and the resulting content of the field at the time the focus was lost.

Taken together, the information contained within these two layers of the Logging module enables a quick and complete reconstruction of a sequence of events as they occurred over time. Importantly, the broader task and user context of these events can also be deduced by exploiting the knowledge of the system and the users embedded within the TIL modules.

3.1.1 Empirical Illustrations

In this section, we present three views illustrating the type of information that can be obtained from the usage logs via querying. These illustrations are based on data from an empirical simulation of our prototype, which we conducted for the purpose of collecting usage data. In this simulation, 15 users employed our TIL-enabled prototype to complete several prototypical ERP tasks during a course of over 27 days. The

logged data included 39 user sessions and 450 different task instances.

Example 1. Table 1 shows an excerpt from the list of task instances in chronological order by their start time, obtained by querying the TIL model data.

The first five columns in this figure describe the user who initiated the task instance, the id of the task instance (TIID), the start and end time, which refer to the time the task was initiated and completed, and the task name. Also included in the view is the output object for each task instance, specified in columns *DTableOut* and *OutPKValue*. For example, the outcome of task instance 111 is a record in the Purchase_Requisition table with the primary key value of 12. The last column, *ParentTIID*, is filled only for editing tasks, as it connects the instance of an editing task to the task instance that first produced the edited object, as shown for task instances 196 and 197.

The excerpt from the log data demonstrates some of the complexities involved in aggregating this data into demonstration scripts:

- An example of a task instance that extended over more than one user session is presented by task instance 111. This Add Purchase Requisition task started on Dec 13 and ended seven days later, when Edit Purchase Requisition number 196 was completed as the result of user3 submitting the requisition. Importantly, our usage log captures the relationship between Add and Edit task instances, enabling the reconstruction of the process of creating a Purchase Requisition from its inception to its submission.
- Parallel execution of multiple task instances by a single user is demonstrated by the timing of task instances 116 and 117, performed by user4. In determining which tasks are related to each other in a business process, the TIL model relies on the object flow between the tasks. The object flow information enables our playback procedure to determine automatically if two co-occurring task instances are related to each other, and, thus, must be shown together in a task tutorial.

  In the case of task instance 116 (Add Purchase Order) and 117 (Review Purchase Requisition), additional querying that is not shown in table 1 can reveal if task instance 117 involved reviewing a purchase requisition that was used as input to task instance 116. Based on that information, our playback algorithm would either include or leave out task instance 117 as a part of the playback of task instance 116.

Thus, the task instance data captured by our TIL-enabled prototype contains the complete chronology of the task instances, including the precise determination of the task instance boundaries in the presence of co-occurring tasks, which is critical for creating task instance visualizations based on usage log data. The logging data also associates each task instance with the resulting changes in the domain data. This allows us to reason effectively about the relationships between the task instances based on the flow of the domain objects.

Example 2. Table 2 details the system-user interactions occurring within a particular task instance. This view aggregates the data from the Logging module so we can trace what happened. It is a chronologically ordered set of all logged user interaction events that occurred while user3 was working on an Add Purchase Requisition task (task instance 111).

As shown in rows 1 through 4, the user entered values in Vendor, Plant, and Storage Location fields on page number 1 of the task interface using the keyboard and then pressed the Submit button. The user subsequently entered the second page of the task interface, where she entered values into fields and saved the resulting purchase requisition for further editing.

In addition to obtaining the details of the input events, we can determine the full set of actual domain input parameters for each task instance by computing the final values in the entry fields of each task page. This is demonstrated by Table 3 for task instance 111. Note that the Quantity values entered for task instance 111 are deliberately left out of Table 3, because quantity is a *non-domain* value, i.e., a number that is not associated with any domain database record. The task instance input and output information plays the key role in determining if task instances are related to each other and in reconstructing process instances for presenting to the user, as described next.

3.2 From Individual Tasks to Processes

The flow of business objects between tasks forms a natural basis for our representation of processes. In the course of system use, objects produced as the output of some tasks are used as inputs to others. We therefore define a process in our framework as consisting of a set of tasks related via their inputs and outputs and associated with the type of object that is ultimately produced.

Our system helps the user in identifying those tasks that are relevant to a process, but the user makes the final selection of which tasks comprise a process at the system configuration stage. This specification is stored in the Task module, while the business ob-

Table 1: Task instance logging.

User	TIID	StartTime	EndTime	TaskName	DTableOut	OutPKValue	ParentTIID
				. . .			
user3	110	13-Dec 12:17:31	13-Dec 12:23:48	Select Task			
	111	13-Dec 12:17:53	20-Dec 15:07:30	Add Purchase Requisition	Puchase_requisition	12	
	112	13-Dec 12:19:32	13-Dec 12:22:12	Add Purchase Order	Purchase_Order	10	
	113	13-Dec 12:22:24	13-Dec 12:23:39	Add Purchase Order	Purchase_Order	11	
user4	115	13-Dec 13:13:08	13-Dec 13:30:27	Select Task			
	116	13-Dec 13:13:35	13-Dec 13:17:38	Add Purchase Order	Purchase_Order	12	
	117	13-Dec 13:15:13	13-Dec 13:15:34	Review Purchase Req			
				. . .			
user3	194	20-Dec 15:04:14	20-Dec 15:04:51	Add Material	Material	70	
	195	20-Dec 15:05:07	20-Dec 15:06:38	Add Goods Receipt	Goods_receipt	6	
	196	20-Dec 15:06:46	20-Dec 15:07:30	Edit Purchase Requisition	Purchase_requisition	12	111
	197	20-Dec 15:07:34	20-Dec 15:08:25	Edit Purchase Requisition	Purchase_requisition	16	126
				. . .			

Table 2: Interface-level details of user actions within a specific task instance derived from the log.

	Task Instance 111, Add Purchase Requisition, by user3						
	User Entry Time	EFID	ICName	PageNum	ValueStart	Entered	ValueEnd
1	13-Dec 12:17:53	1393	Vendor	1		2	2
2	13-Dec 12:18:08	1394	Plant	1		3	3
3	13-Dec 12:18:20	1396	Storage Location	1		2	2
4	13-Dec 12:18:32	1401	Standard Form Button	1	Submit	<press>Submit	Submit
5	13-Dec 12:18:36	1404	Material	2		<press>Lookup	
6	13-Dec 12:18:41	1404	Material	2		<press>Select	
7	13-Dec 12:18:41	1404	Material	2		<receiveSelection>	22
8	13-Dec 12:18:42	1405	Quantity	2	1	2	2
9	13-Dec 12:18:48	1416	Standard Form Button	2	Add Line	<press>Add Line	Add Line
10	13-Dec 12:18:48	1417	Material	2		<press>Lookup	
11	13-Dec 12:18:54	1417	Material	2		<press>Select	
12	13-Dec 12:18:54	1417	Material	2		<receiveSelection>	64
13	13-Dec 12:18:55	1418	Quantity	2	1	5	5
14	13-Dec 12:19:05	1414	Standard Form Button	2	Save	<press>Save	Save

ject that is produced by the process is stored in the Domain module.

Given a process specification as a set of tasks connected via output/input links, *process instances* can be reconstructed automatically from the Logging module within the TIL model. The algorithm for process instance detection is implemented as an SQL procedure over the TIL and Domain data and is best described as a breadth-first search in the *task instance graph*.

A task instance graph is a directed acyclic graph, in which the nodes correspond to task instances, and an arrow from node a to node b is drawn if the output object of task instance a was used as input to task instance b. Figure 3 depicts a task instance graph in which one process instance is highlighted in boldface.

The input to the process instance identification algorithm is a specific business object, e.g., good receipt (or GR) #34, and a set of tasks comprising the process, e.g., $P = \{a,b,c,g,e,f\}$, as described in the Task module. The output is a set of task instances comprising the process instance that produced the specified business object. Throughout its operation, the algorithm considers only instances of tasks in the process task set P. It starts by identifying the chronologically latest task instance that produced the specified object as its output and putting that instance in a list of discovered task instances D. The breadth-first traversal then starts from that latest task instance.

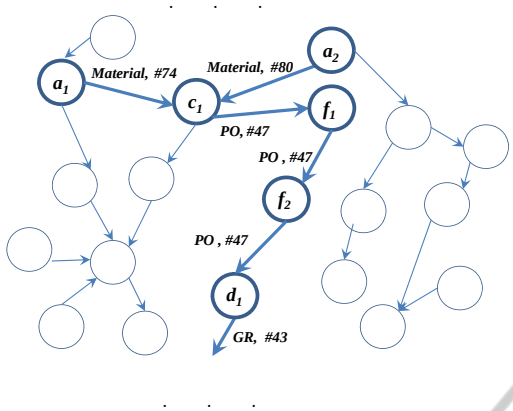

Figure 3: A fragment of a task instance graph with a process instance highlighted in boldface.

At each subsequent stage, for each node $\delta \in D$, the search identifies the set of all nodes with links into δ that preceded it in time and adds it to D. The search continues until no new nodes are discovered. The process instance is returned as the set of task instances D.

Note that, due to the optionality of some tasks within a process, there can be great variation in the actual task composition of process instances. Another source of variability is the timing of the task instances involved in a process. Our approach to process instance detection accurately identifies process instances using a SQL-based procedure over the TIL model data, thereby avoiding the pitfalls typically encountered by data mining approaches.

To summarize, the process-related capabilities of our current system prototype include:

Process Configuration Stage: For a given output document type, the user specifies the set of tasks that she wants to have included in the designated process. The tasks must be related to each other via their input-output links (Lucas and Babaian, 2012).

At System Runtime: When a user selects the playback option on a task, one of the choices includes the demonstration of a process involving that task. If the person chooses to play the process, the system queries its log to produce a list of existing completed process instances and presents the choices to the user. The selected instance is demonstrated to the user by the system in the Playback window, as described next.

Table 3: Domain objects used as input to task instance 111.

User	TIID	DTable	PKValue Entered
user3	111	Material	22
		Material	64
		Plant	3
		Storage_loc	2
		Vendor	2

4 REPLAYING THE INTERACTION

In this section, we describe our interface and algorithm for creating on-demand playbacks of tasks. We have implemented and tested the interface within our proof-of concept prototype ERP system.

For every available task in the prototype, the user can invoke a ShowMe option, which demonstrates the task interface by playing back a previously recorded system-user interaction of that task. The visualization is performed by the Playback procedure, described in the next subsection, and involves imitation of the recorded user actions and system responses at a speed that makes it easy to follow the process.

Upon invoking the ShowMe option, the user can select demonstrations displaying a task instance that was:

- performed by a particular colleague,
- completed during a specified range of dates, or
- involved a specific business object, e.g., customer invoice number 254.

We chose these parameters because they most closely correspond to the types of information being sought by users in real-life scenarios.

Another playback option is the demonstration of an entire process involving a particular task, as many ERP users in our field studies complained about not understanding where a task fits into an overall process in terms of what tasks preceded it and what tasks followed it. Understanding how tasks are linked and the flow of data through a process is often essential information, particularly when users attempt to address an error caused by a problem with the data, but this information is not readily available from the system.

Figure 4 shows a screenshot of a Playback Parameter Selection window. In this case, the user has specified that they would like to see how user11 has performed the Add Purchase Requisition task within the specified time range. The user selected to view the task within the context of the corresponding Purchase Order process instance.

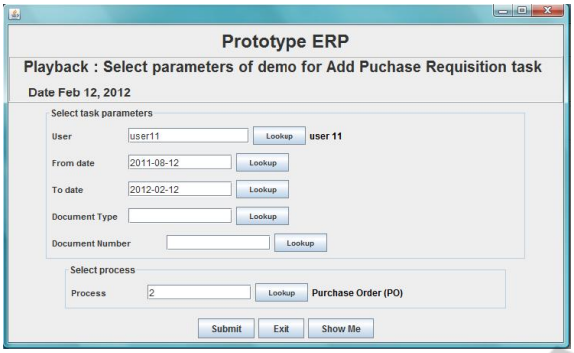

Figure 4: A screenshot showing the parameters that can be specified for the selection of the task playback.

In response to a ShowMe request, the system queries the Logging module for a list of task or process instances matching that request and presents them to the user. If no parameters are specified, the system generates a list of all available instances of the requested task. At this time, the list of playable task and process instances is not rank-ordered in any way. In the future, we plan to investigate effective ways of utilizing the wealth of information available in the logs about the behavior of the user making the request as well as others performing similar tasks for use in rank ordering the choices to best fit the user's profile and goals.

4.1 The Playback Procedure

The pseudocode for the Playback procedure, which is run by the system after the user selects a task instance for demonstration, is depicted in Figure 5. The procedure is passed a task instance identifier *ti* and the page number within the task interface from which to start the playback. The procedure starts by querying the Task and Interface modules for the composition of the task page, which is required for rendering it (see the call to *getTaskPageInterface* function in line 1). In the next step, the system retrieves the records of user activities for this task instance on this task page from the log in chronological order. We call this set of records a *script*, as it contains the details needed for playing back the task instance. Table 2 depicts the essential columns of the playback script. In step 3, the initial values of the input fields in the task page at the time of *ti* are determined based on the *script*; the page is then displayed in a separate Playback window.

For the effectiveness of the demo, the playback script is pre-processed in step 5 to remove any obviously ineffectual key/mouse press sequences. For example, consider the situation in which a user, having invoked a lookup interface, presses the "Display All" button (shown in Figure 1) more than once. None of the button presses after the first one lead to a change in the state of the system and are therefore eliminated from the playback. Another example is a canceled lookup action, which would also add no value.

Lines 6-20 of the pseudocode specify a loop in which every record of the *script* is visualized. Recall that each record contains information on the invoked input component. The record is first analyzed to determine what kind of input component is being referenced. In the case where the record refers to interaction with an input field:

- if the value was typed into the field, as specified, for example, by the log record in row 1 in Table 2 for the Vendor value, the *showExecution*(*record*) function displays the keyed value being entered directly into the text field, or
- if the lookup functionality was invoked (see log records displayed in rows 5-7 in Table 2), the *showExecution*(*record*) function shows the interaction occurring within the lookup interface followed by the transfer of the value selected from the list into the text field.

In the case where the record refers to a button press: the *showExecution*(*record*) function visualizes the button press and its visible effects, such as any informational prompts that occur in response to the press event. If the Submit button was pressed, the Playback procedure is called recursively to visualize the interaction on the next page of the task instance (lines 11-13).

Since menu items in the interface provide quick access to tasks that are closely related to the current one, invoking a menu item triggers another call to the Playback procedure. First, however, which task interface was enacted (*relatedTask* in line 16) must be determined by consulting the Interface and Task modules and, from that information, finding the corresponding task instance (*relatedTI* in line 17). The *relatedTI* task instance is identified by the *findNextTI*() function as the one occurring closest in time within the current user session to the invocation of the menu instance. If that task instance did not involve any objects used as the input to the currently visualized task instance *ti*, it is discarded. Otherwise, the identified *relatedTI* is visualized in a separate window via the call to Playback (*relatedTI*,1) in line 18.

5 CONCLUSIONS AND FUTURE WORK

We have demonstrated how usage histories collected by the system can be put to use for automati-

Procedure: Playback.
Input: task instance ti, starting page number *pageNum*
Output: a dynamic visualization of the step-by-step execution of ti

```
1   taskPage = getTaskPageInterface(ti,pageNum)
2   script = getTIScriptFromLogs(ti,pageNum)
3   initializeTaskPageInputFields (taskPage,script)
4   display(taskPage)
5   remove records of ineffectual actions from script
6   for each record in script
7       if (inputControlType(record) == INPUT_FIELD )
8           showExecution(record)
9       else if (inputControlType(record) == BUTTON)
10          showExecution(record)
11          if (getButtonType(record) == SUBMIT_BUTTON and
12              pageNum != lastPage(ti))
13              Playback(ti,pageNum + 1)
14          end if
15      else if (inputControlType(record) == MENU_ITEM)
16          relatedTask = taskInvokedViaMenu (record)
17          relatedTI = findNextTI(task,sessionID,record.StartTime)
18          if (relatedTI != NULL) Playback(relatedTI,1) endif
19      endif
20  end for each
```

Figure 5: Pseudocode for procedure Playback.

cally creating demonstrations of the system interface as dynamic visualizations of previously recorded system-user interactions. We have outlined the Task-Interface-Log (TIL) data model that is used to support such demonstrations and discussed the components that are critical to enabling their efficient, on-demand composition from key-press-level and task-level log data. The prototype we described is implemented in Java and MySQL.

A number of simplifications in our prototype have been made; notably, the number of tasks and variety of input components are more limited than those found in modern ERP systems. Our intention is not to replicate the scope and scale of these systems but rather to demonstrate the feasibility of our approach and test the design of its critical components, i.e., the TIL model and the algorithms for on-demand task and process instance identification and dynamic playback. In the future, we plan to work on user-testing and fine-tuning our initial proof-of-concept version of the playback interface presented here.

Given the complexity of ERP system interfaces, the ability to replay previously occurring interactions is beneficial to users in a variety of circumstances, including new users performing a process for the first time, more experienced users performing particular tasks on an infrequent basis, or any user who just does not remember how to proceed. Building the ability to dynamically generate automated playbacks of user-system interactions right into the system is also a more efficient and lower cost option than having users rely on others within the organization for help.

In our framework, processes are formed by following the logical flow of business objects between tasks, with the system helping the user identify those tasks that are relevant to each process. Defined in this way, the concept of a process is flexible, yet process instances can be effectively and efficiently reconstructed from the logs. The rich data set of past interactions can also be exploited for rank-ordering demonstrations to best fit the user's needs. Additional applications of log data analysis to user support enabled by our representation include providing the user with recommendations regarding the next task to perform in a process, determining the user with the most experience in performing a particular task, computing a list of incomplete tasks for a user, enabling annotations and user-editing of tutorial playback scripts, and auditing the usage history for compliance with process guidelines. We are exploring several of these possibilities in furthering our long-term research goal of making organizational systems function as their users' helpful and intelligent partners.

ACKNOWLEDGEMENTS

This material is based in part upon work supported by

the National Science Foundation under Grant No. 0819333. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

Brusilovsky, P. and Cooper, D. W. (2002). Domain, task, and user models for an adaptive hypermedia performance support system. In *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces*, pages 23–30, New York, NY, USA. ACM Press.

Caldwell, D. E. and White, M. (1997). Cogenthelp: a tool for authoring dynamically generated help for java guis. In *Proceedings of the 15th annual international conference on Computer documentation*, SIGDOC '97, pages 17–22, New York, NY, USA. ACM.

Chakravarthi, Y. A., Lutteroth, C., and Weber, G. (2009). Aimhelp: generating help for gui applications automatically. In *Proceedings of the 10th International Conference NZ Chapter of the ACM's Special Interest Group on Human-Computer Interaction*, CHINZ '09, pages 21–28, New York, NY, USA. ACM.

Cooprider, J., Topi, H., Xu, J., Dias, M., Babaian, T., and Lucas, W. (2010). A collaboration model for erp user-system interaction. In *Proceedings of HICCS'2010*, pages 1–9. IEEE Computer Society.

Da Cruz, A. M. R. and Faria, J. a. P. (2010). A metamodel-based approach for automatic user interface generation. In *Proceedings of the 13th international conference on Model driven engineering languages and systems: Part I*, MODELS'10, pages 256–270, Berlin, Heidelberg. Springer-Verlag.

Dorn, C., Burkhart, T., Werth, D., and Dustdar, S. (2010). Self-adjusting recommendations for people-driven adhoc processes. In *Proc. of the 8th international conference on Business process management*, BPM'10, pages 327–342. Springer-Verlag.

Greco, G., Guzzo, A., and Sacc, D. (2005). Mining and reasoning on workflows. *IEEE Transaction on Knowledge and Data Engineering*, 17:2005.

Grossman, T. and Fitzmaurice, G. (2010). Toolclips: an investigation of contextual video assistance for functionality understanding. In *Proc. of the 28th international conference on Human factors in computing systems*, CHI '10, pages 1515–1524. ACM.

Hamerman, P. (2007). ERP applications 2007: Innovation rekindles. Forrester Research.

Hestermann, C. (2009). Key issues for enterprise resource planning. Gartner.

Iansiti, M. (2007). Erp end-user business productivity: A field study of sap & microsoft. Technical report, Keystone Strategy.

Ivory, M. Y. and Hearst, M. A. (2001). The state of the art in automating usability evaluation of user interfaces. *ACM Computing Surveys*, 33(4):470–516.

Leshed, G., Haber, E. M., Matthews, T., and Lau, T. (2008). Coscripter: automating & sharing how-to knowledge in the enterprise. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1719–1728, New York, NY, USA. ACM.

Linton, F., Joy, D., Schaefer, H.-P., and Charron, A. (2000). Owl: A recommender system for organization-wide learning. *Educational Technology & Society*, 3(1).

Lucas, W. and Babaian, T. (2012). Implementing design principles for collaborative ERP systems. In *Proc. of the 7th International Conference on Design Science Research in Information Systems and Technology, DESRIST'12, Las Vegas, NV, May 2012.*, Lecture Notes in Computer Science, pages 88–107. Springer.

Mahfoudhi, A., Abid, M., and Abed, M. (2005). Towards a user interface generation approach based on object oriented design and task model. In *Proc. of the 4th international workshop on Task models and diagrams*, TAMODIA '05, pages 135–142. ACM.

Plaisant, C. and Shneiderman, B. (2005). Show me! guidelines for producing recorded demonstrations. In *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 171–178. IEEE Computer Society.

Ramachandran, A. and Young, R. M. (2005). Providing intelligent help across applications in dynamic user and environment contexts. In *Proceedings of the 10th international conference on Intelligent user interfaces*, IUI '05, pages 269–271, New York, NY, USA. ACM.

Rozinat, A. and Van der Aalst, W. M. P. (2008). Conformance checking of processes based on monitoring real behavior. *Inf. Syst.*, 33(1):64–95.

Shen, J., Fitzhenry, E., and Dietterich, T. G. (2009). Discovering frequent work procedures from resource connections. In *Proceedings of the 13th international conference on Intelligent User Interfaces*, pages 277–285.

Topi, H., Lucas, W., and Babaian, T. (2005). Identifying usability issues with an ERP implementation. In *Proc. of the International Conference on Enterprise Information Systems (ICEIS-2005)*, pages 128–133.

Tran, V., Kolp, M., Vanderdonckt, J., Wautelet, Y., and Faulkner, S. (2010). Agent-based user interface generation from combined task, context and domain models. In England, D., Palanque, P. A., Vanderdonckt, J., and Wild, P. J., editors, *TAMODIA*, volume 5963 of *Lecture Notes in Computer Science*, pages 146–161. Springer.

Van der Aalst, W. M. P. (2010). Process discovery: Capturing the invisible. *IEEE Comp. Int. Mag.*, 5(1):28–41.

Van der Aalst, W. M. P., Pesic, M., and Song, M. (2010). Beyond process mining: From the past to present and future. In *CAiSE 2010*, pages 38–52.

Yeh, T., Chang, T.-H., Xie, B., Walsh, G., Watkins, I., Wongsuphasawat, K., Huang, M., Davis, L. S., and Bederson, B. B. (2011). Creating contextual help for guis using screenshots. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pages 145–154, New York, NY, USA. ACM.