Implementing Design Principles for Collaborative ERP Systems[⋆]

Wendy Lucas and Tamara Babaian

Bentley University, Waltham, MA 02452, USA
{wlucas,tbabaian}@bentley.edu

Abstract. Enterprise Resource Planning (ERP) Systems are notoriously difficult for users to operate. We present a framework that consists of a data model and algorithms that serve as a foundation for implementing design principles presented in an earlier paper for improving ERP usability. The framework addresses the need for providing user, task and process context of each system-user interaction. It is intended to form an integral part of the system's data model, which can be queried in real time to produce the information required for a variety of user interface enhancements. We have implemented the framework within an ERP prototype and used it in a laboratory emulation of ERP usage. Using the log data from this laboratory emulation, we present examples demonstrating how the framework meets its design goal of providing contextual and historical information.

Keywords: Usability, human-computer collaboration, enterprise systems, ERP, human-computer interaction.

1 Introduction and Motivation

Enterprise Resource Planning (ERP) systems integrate data and information flow from throughout the organization. Companies rely on them for standardizing their processes around best practices. Rather than the system conforming to the way a particular company does business, the company must conform to the system-prescribed approach in order to reap the maximum benefit. Representing industry-wide rather than company-specific practices places a heavy burden on the user, who must undergo extensive training to learn how to perform particular tasks with the system. Users typically memorize how to do those tasks, as the underlying processes are hidden behind very complex interfaces and little guidance or support is provided by the system. The poor usability of ERP systems has been noted in industry reports [16, 22, 15, 17] and our own field studies [26, 10], yet usability problems still abound. Considerable advances in research on

⋆ 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.

human-computer interaction [19] have also not resulted in significant improvements in ERP system design.

The work presented here is part of a comprehensive research effort aimed at achieving a breakthrough in the usability of enterprise systems by applying the human-computer collaboration paradigm [25] to system design and evaluation. This paradigm is grounded in theory of collaboration and requires that the system act as a partner that supports its users in the increasingly complex environments of modern applications [13]. To be a collaborative partner, the system must do its part by sharing information and adjusting its behaviors based on its knowledge and awareness of the user, the context of the interaction, and its own functionality. (Note that this is different from Computer-Supported Cooperative Work (CSCW), which is concerned with computing technology that supports human collaboration).

In previous work [7], we derived four design principles based on system-user collaboration for addressing the usability issues identified in our field studies. In this paper, we present a representational framework and algorithms that serve as a foundation for implementing these principles. In validating our approach, we focus here on two of these principles, which are referred to as Design Principle 2 (DP2) and Design Principle 4 (DP4). DP2 concerns providing context- and user-appropriate navigational and progress guidance to the user. DP4 focuses on improving access to data and actions that are most likely to be relevant and useful. The other two principles, which involve mechanisms for customization and error handling, are also supported by our framework and are topics of ongoing research that is beyond the scope of this paper.

Our framework specifies a model that represents the system's task structure, interface components, and usage log of all user-system interactions. It has been specifically designed to enable the system to make effective use of usage histories during system-user interactions. This model, which we refer to as the Task-Interface-Log, or TIL, model, provides the requisite information for supporting the design principles by explicitly associating low-level user inputs with higher-order processes. Our approach is further distinguished by its use of logged data in support of system-user interactions in real time, as opposed to the off-line processing of logged data for process mining and discovery purposes [2, 23], which is the more common focus of research involving usage logs in the enterprise system domain.

We have implemented the TIL model in SQL and embedded it in an ERP prototype. To evaluate the capabilities of both the TIL model and the algorithms for supporting the design principles, we conducted an emulation of the use of ERP systems in a laboratory setting. The usage data collected from this emulation was used to validate our approach.

The next section of this paper presents related work. This is followed by descriptions of the design principles. We then present our representational framework and examples that illustrate the framework's utility using empirical data. We conclude with a discussion and directions for future work.

2 Related Work

Usage data has been used extensively for extending the functionality of automated tutoring, recommender, and adaptive hypermedia applications (e.g. [21, 9]). Jameson [20] provides a review of interfaces that adapt their behaviors in order to better meet the needs of the user. In these applications, the possible set of actions that a user can perform is typically well-defined, and the emphasis is on modeling the user in order to provide suitable recommendations, guidance, and support.

The ability to reason from usage logs for supporting users in real time within the context of complex enterprise systems is far less commonplace. Günther et al. [14] describe how event logs are recorded at very low levels of abstraction, making them difficult to relate to activities within a process. As noted by Ivory and Hearst [18], keystroke data is easy to record but difficult to interpret. Our framework overcomes this hurdle by associating interface components with both contextual information and usage histories, making it possible to analyze and utilize data ranging from the keystroke level to the task level, from a single user to multiple users.

While ERP users are constrained by the business logic of the system, there are no strictly enforced process models. Rozinat and Aalst [23] have shown that activities followed in completing a process, as mined from ERP system logs and other administrative systems, often deviate from the prescribed process. A variety of algorithms and techniques exist for constructing process models from low-level events in a usage log [2]. Investigating system usage by applying such techniques for deriving workflow models that can then be analyzed off-line is the focus of much of the work in this area (see, for example, [4, 5, 11, 6]).

Our own interest lies in developing design models that enable a system to provide dynamic guidance and support to users based on process sequences corresponding to actual organizational practices, which is the subject of far less research. Aalst et al. [3] describe an application that focuses on processes that have not yet completed for checking on conformance, estimating completion times, and making recommendations on steps for minimizing overall flow times. Schnonenberg et al. [24] propose a recommendation service that guides users by giving recommendation on possible next steps based on past process executions. It has been implemented in ProM [4], an open-source process mining framework for implementing process mining tools. A partial case from the user who is seeking guidance, consisting of a sequence of performed steps, is sent to to the recommendation service. The case is then compared to the log, and a recommendation is generated.

The above works are all based on discovering processes from usage log activity traces that are contained within the time period between the predefined starting and ending activities. This approach is complicated by the noise that comes from the non-process related tasks that are commonly interleaved with the process-related ones within the identified time period. Aalst [1] notes that there are several shortcomings to existing algorithms due to processes being highly concurrent and the existence of complex dependencies between activities.

In our framework, we avoid many of the challenges inherent in mining-based approaches because our model contains the specification of tasks included in the process. Our approach is distinguished by the ability to automatically and accurately identify process instances based on log records, by virtue of the direct representation of tasks, processes and the flow of domain objects in the TIL model.

3 Design Principles

Given the integrated nature of ERP systems and the complexity of their design, approaches that specify isolated patches for addressing particular issues will not succeed in improving overall system usability. Rather, a systematic approach for evaluating and addressing usability issues is required. Our field studies of ERP system users revealed common categories of usability issues that can be explained as examples of non-collaborative behavior between the system and its users. We applied collaboration theory [8, 13] as the unifying perspective for viewing human-computer interactions in deriving our design principles [7], which are presented in figure 1. What we refer to throughout this paper as *processes* are referred to as *transactions* in these principles.

DP1 The user interface should provide a mechanism for customizing the vocabulary of terms used by the system in its communication to the user, the composition of business transactions, and the content of the system's informational output to match the practices of the organization. There should be a mechanism for incorporating the customizations from an earlier version of the system to a later one.

DP2 The system should provide navigational and progress guidance to a user performing a transaction, indicating the broader context of each interaction in terms of the related business process components and specifying the completed and remaining parts. A sufficiently competent user should be able to turn off this guidance if it becomes a distraction.

DP3 When the system detects a problem, it should identify the possible causes and ways of resolving it. If the fix is obvious, the system should inform the user and perform it. If it isn't obvious, the possible causes and resolution scenarios should be presented to the user and be readily executable. If the system is unable to identify resolution strategies, it should present the user with the relevant data and transactions.

DP4 In presenting selection choices, the system should utilize what it knows about the user, the organization, the task, and the context, and provide faster access to the more likely choices than the less likely ones. Where the choice of data or action is obvious, the system should have an option of not waiting for the user to enact it. The user should have an option to replace/cancel the system's provided choice of data/action.

Fig. 1. Design principles for greater ERP usability

DP1 grew out of reported instances of users needing to undergo a lengthy process, characterized by some as “brutal” and “intimidating,” of learning the language of the system and adapting to its practices. DP2 arose from the difficulties users face in understanding the process flow and navigating the system, with little support on how to proceed or what progress has been made. DP1 and DP2 are meant to address the failure of the system to be a good collaborative partner by *communicating* its knowledge concerning the steps that need to be taken, the means for performing them, and the progress made toward achieving the goal.

Numerous reports by users of their inability to determine the cause of an error, decipher error messages, or figure out how to address the problem led to the statement of DP3. In such cases, the system is failing to *help* a partner in need. Lastly, DP4 grew out of observed and reported cases of the system presenting all possible choices, even those that will not work in the current situation, in search interfaces, lists, etc., and failing to take the user’s previous entries and actions into account. In these cases, the system has not provided appropriate *support* to assist the user in daily operations.

The framework presented in this paper provides the information needed for supporting all four principles. We have limited our validating examples in the next section to two of these, DP2 and DP4, due to space limitations. Examples related to DP1 and DP3 will be presented in future work.

4 Representational Framework

In this section, we present the representational framework that we developed to support implementation of the design principles. The design goals behind this framework originated from the requirements on the system’s awareness of historical and contextual data, as necessitated by the design principles:

1. to represent the system’s task and interface structure in a way that enables reasoning about their relationship to each other and to the ERP domain data in the context of a business process,
2. to capture and store the history of each system-user interaction in a way that enables a quick identification of the task and user context of all past and on-going interactions, as well as recording the lower-level keyboard and mouse input details, and
3. to make the knowledge included in the first two items accessible to the system at run-time for supporting a variety of implementations of the design principles.

The framework includes the Task-Interface-Log (TIL) data model, algorithms for deriving process-related data, and input-aware components. The following sections describe the model and algorithms. While the components have also been implemented, they are not reviewed here.

4.1 Overview of the TIL modules: Task, Interface, Logging

At the core of our representational framework is the TIL data model for representing Tasks and their inclusion in business processes, the Interface components that implement them, and usage Logs that store the details of system-user interactions.

The *Task* module of the TIL model captures the description of tasks that the system implements and their inclusion in business processes. The set of interface pages associated with each task is described in the *Interface* module. This module also describes the composition of each interface page from user input controls, such as input fields, buttons, and menus. The descriptions within the Task and Interface modules are *static*, in that they do not change with use of the system, with one exception that allows the system to be configured with business process specifications as desired collections of tasks. The data within these two modules is used to render interface pages, when the user invokes a task interface, and for tracking the task and process context of each interaction.

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 user's engagement with the system on a particular task. A task instance can extend over multiple user-sessions, and the Logging module chronicles the execution of a task instance from the beginning to the end.

The interface layer log stores the detailed key-press level information regarding the user's interactions with input controls within the task. To support usage data capture, we have implemented and used a library of user input components that record the interaction data. 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.

The records of organizational data, such as customers, vendors, and invoices, are stored in the ERP system database, which we refer to as the *Domain* module. We call Domain module entities *domain objects* and their corresponding tables *object types.*

Definition 1. *A* domain object *is a record from a table in the Domain module. The* domain object type*, or simply* object type*, is the name of the table in the Domain module storing the domain object.*

All three modules of the TIL model are also related to the Domain module - these relationships specify the flow of organizational data through the tasks. In particular, each task description (Task module) includes a specification of the type of organizational data object that the task produces, called the task's *output object type*. For example, the *Add Material* task produces a record in the *Material* table; thus, it's output type is *Material.* The references to the actual objects (e.g. a concrete *Material* record) produced as a result of a specific task instance are contained within the Logging module's record of task instances.

Along with the output object type for a task, the TIL model also includes information on each task's *input object types.* Each user input component description in the Interface module specifies the type of object that should be entered in the field. Since each input component is associated with a task, the TIL model enables the derivation of the set of input objects used in a task.

4.2 TIL relations

This section introduces the details of the TIL model that are essential to the algorithms and derivations that follow. Boldface is used to denote the names of the relations, and italics is used for the attributes. The relations are defined over standard SQL types, such as varchar, int and datetime. Please refer to figure 2 for descriptions of the attributes of each relation that we review below.

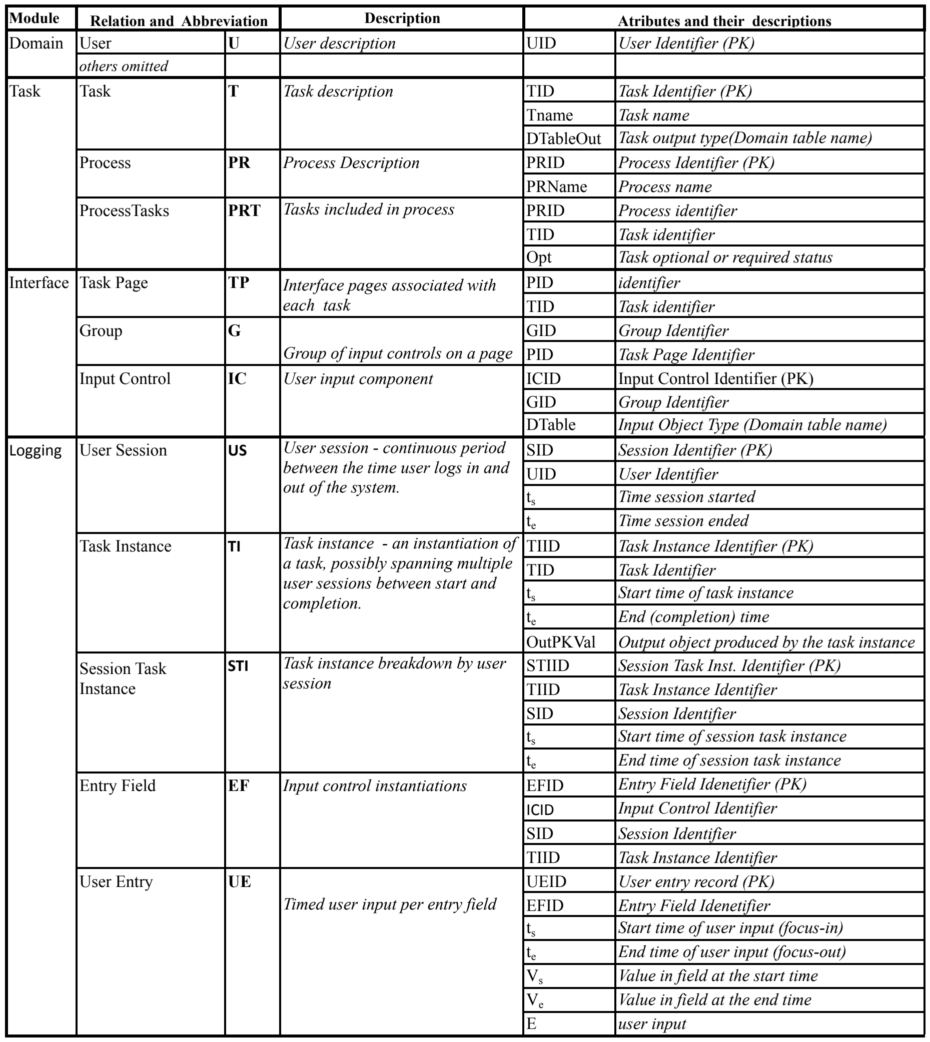
The Domain module represents the ERP organizational data and is not part of TIL, but TIL model relations reference the Domain module tables that store a variety of domain objects. The **User** relation of the Domain module has a special significance, because it is linked to all usage log related records. For the sake of simplicity, we model the **User** relation as consisting of the single identifier attribute *UID*. Other attributes describing the user's relationship with a particular organizational unit, role, or set of permissions within the system could be added for greater richness of informational queries from the log data, but such treatment of **User** is not included in this paper.

The Task module consists of three relations: **Task**, **Process** and **ProcessTasks**. The *DTableOut* attribute of the **Task** table refers to the task's output type which, as we defined in section 4.1, is the name of a table from the Domain module that stores the output objects associated with that task. The **Process** and **ProcessTasks** tables list the business processes and specify which tasks are included in each process, respectively.

The Interface module represents user interface components and their organization and relationship with the Task and Domain modules. Each **Task** is associated with a set of distinct **TaskPages** which, in turn, consist of **Groups** of **InputControls**.

The **InputControl** table describes interactive GUI elements such as buttons, text fields, lists, and menu items. Input control records for text fields specify in the *DTable* column the type of object that must be entered into the text field. For example, a field designated for a customer number will have the *DTable* attribute value equal to **Customer**, which is the table storing customer information in the Domain module. The *DTable* column of input controls used for entering non-domain object data, such as an order quantity or a delivery date, has no value.

The Logging module records the usage history and describes the relationships between click-level and keyboard-level data, users, and tasks.



Module	Relation and Abbreviation		Description	Atributes and their descriptions	
Domain	User	**U**	*User description*	UID	*User Identifier (PK)*
	others omitted				
Task	Task	**T**	*Task description*	TID	*Task Identifier (PK)*
				Tname	*Task name*
				DTableOut	*Task output type(Domain table name)*
	Process	**PR**	*Process Description*	PRID	*Process Identifier (PK)*
				PRName	*Process name*
	ProcessTasks	**PRT**	*Tasks included in process*	PRID	*Process identifier*
				TID	*Task identifier*
				Opt	*Task optional or required status*
Interface	Task Page	**TP**	*Interface pages associated with each task*	PID	*identifier*
				TID	*Task identifier*
	Group	**G**	*Group of input controls on a page*	GID	*Group Identifier*
				PID	*Task Page Identifier*
	Input Control	**IC**	*User input component*	ICID	Input Control Identifier (PK)
				GID	*Group Identifier*
				DTable	*Input Object Type (Domain table name)*
Logging	User Session	**US**	*User session - continuous period between the time user logs in and out of the system.*	SID	*Session Identifier (PK)*
				UID	*User Identifier*
				t_s	*Time session started*
				t_e	*Time session ended*
	Task Instance	**TI**	*Task instance - an instantiation of a task, possibly spanning multiple user sessions between start and completion.*	TIID	*Task Instance Identifier (PK)*
				TID	*Task Identifier*
				t_s	*Start time of task instance*
				t_e	*End (completion) time*
				OutPKVal	*Output object produced by the task instance*
	Session Task Instance	**STI**	*Task instance breakdown by user session*	STIID	*Session Task Inst. Identifier (PK)*
				TIID	*Task Instance Identifier*
				SID	*Session Identifier*
				t_s	*Start time of session task instance*
				t_e	*End time of session task instance*
	Entry Field	**EF**	*Input control instantiations*	EFID	*Entry Field Idenetifier (PK)*
				ICID	*Input Control Identifier*
				SID	*Session Identifier*
				TIID	*Task Instance Identifier*
	User Entry	**UE**	*Timed user input per entry field*	UEID	*User entry record (PK)*
				EFID	*Entry Field Idenetifier*
				t_s	*Start time of user input (focus-in)*
				t_e	*End time of user input (focus-out)*
				V_s	*Value in field at the start time*
				V_e	*Value in field at the end time*
				E	*user input*

Fig. 2. A summary of the essential components of the TIL model and the Domain module.

The **UserSession** relation represents periods of time during which the user is continuously logged in to the system. It is used to relate each interaction to a particular user.

The **TaskInstance** relation records instantiations of tasks. A new task instance record is created each time the user opens a new task. The task instance end time corresponds to the moment when the user either cancels the task instance or completes it, in which case the output is saved in the Domain database.

The identifier of the output object is stored in the *outPKVal* column of the Task Instance record.

As the task instances can span multiple user sessions, the **SessionTaskInstance** relation is used to record the task instance execution times within different sessions.

Taken together, the **UserSession**, **TaskInstance** and **SessionTaskInstance** relations specify the user and task context of system-user interactions. The detailed log of system-user interactions within the task instances is stored within the **EntryField** and **UserEntry** relations, described next.

The **EntryField** table represents the instantiations of input controls corresponding to a specific task instance and user session. The *ICID* attribute refers to the instantiated input control. *SID* and *TIID* are references to the session and task instances, respectively, in which the entry field was created.

The **UserEntry** relation records the user input directed to the specified entry field. The start and end times of the period when the entry field is in continuous focus are defined by t_s and t_e. Attributes V_s and V_e record the value in the text field at the start and the end of that time period, while E denotes a string recording the user's input as a sequence of keystrokes or mouse events.

4.3 Task and process graphs and algorithms

To provide context-aware guidance and navigational support for design principle DP2 requires that the system be aware of the relationships between the tasks and the input-output flow of objects between them. The TIL model specifies the input and output types of tasks in a process and, during runtime, records the actual domain objects that are instantiated. This information enables the automatic determination of the relationships between task and task instances in a chain comprising a business process. Figure 3 illustrates the types of task and process-related information that we focus on in this section.

Task graph. Figure 3(a) presents a fragment of a system *task graph* that can be composed from descriptions contained within the Task and Interface modules. The nodes correspond to tasks, and an arrow from one node to another designates that the output of the source task may be used as an input to the target task. For example, arrows from task a, Add Material, lead to tasks b, c, e, and f. This is because these four tasks have Material as part of their input, which can be established by querying the TIL model records on the input fields for these tasks.

The task graph is composed from the TIL model data using procedures *DInput* and *DOutput*, which stand for Domain Input and Domain Output and are depicted in figure 4. We use relational algebra operations [12] of natural join ($*$), projection (π), selection (σ) and renaming (ρ). We use the abbreviated names of the TIL relations, as presented in the third column of figure 2. Uppercase letters used here and throughout the paper denote relations, while the names of scalar values and individual tuples begin with a lowercase letter.

Procedure *DInput*(*paramTID*) returns a set of domain input types of task *paramTID*, i.e. the types of domain objects that can be entered as an input to

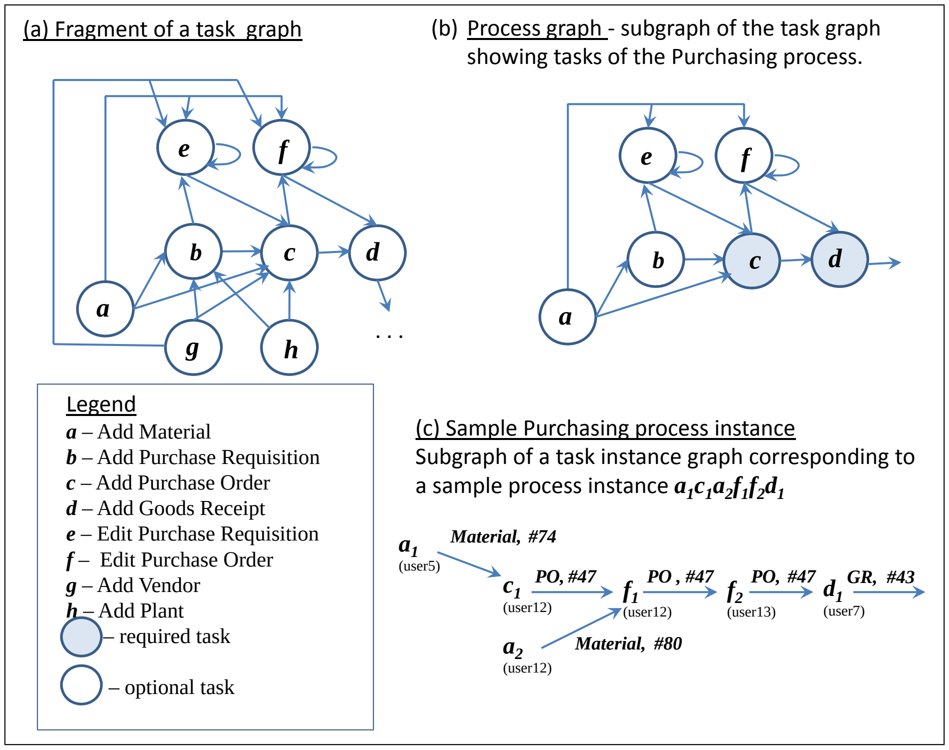

(a) Fragment of a task graph
(b) Process graph - subgraph of the task graph showing tasks of the Purchasing process.
a
b
c
d
e
f
g
h
. . .
Legend
a – Add Material
b – Add Purchase Requisition
c – Add Purchase Order
d – Add Goods Receipt
e – Edit Purchase Requisition
f – Edit Purchase Order
g – Add Vendor
h – Add Plant
– required task
– optional task
(c) Sample Purchasing process instance
Subgraph of a task instance graph corresponding to a sample process instance $a_1c_1a_2f_1f_2d_1$
a_1 (user5)
Material, #74
c_1 (user12)
PO, #47
f_1 (user12)
PO, #47
f_2 (user13)
PO, #47
d_1 (user7)
GR, #43
a_2 (user12)
Material, #80

Fig. 3. Task, process and process instance graphs.

task *paramTID*. (We refer to the tuples from our data model by their identifier value, as described in the fifth column of figure 2.) *DInput* computes the result by collecting the set of table names associated with all of the task's input controls. Procedure *DOutput*(*paramTID*) returns the value of the *DTableOut* attribute, which specifies the output table for the task with identifier *paramTID*.

Definition 2. *A* task graph *is a directed graph in which the set of nodes corresponds to the tasks, and a link from task a to task b is drawn if and only if $DOutput(a) \in DInput(b)$.*

Process graph. In our framework, the processes are specified as a set of tasks, which must be related via their inputs and output. We define business processes as being comprised of one or more required tasks and zero or more optional tasks. Optional tasks are those that are not required by the system to complete a process. Process compositions from tasks can be configured by organizations to match their own practices.

Definition 3. *A* process *is a set of tasks, which form a weakly connected subgraph in the task graph. Some steps in a process are designated as required, while the rest are optional.*

```
Procedure DInput
Input: task id $paramTID$
Output: set of domain input types of task $paramTID$
1 $R = \pi_{\mathbf{IC}.DTable}(\sigma_{\mathbf{T}.TID=paramTID}(\mathbf{T} * \mathbf{TP} * \mathbf{G} * \mathbf{IC}))$
2 return: $R$
```

```
Procedure DOutput
Input: task id $paramTID$
Output: domain output types of task $paramTID$
1 return: $\pi_{\mathbf{T}.DTableOut}(\sigma_{\mathbf{T}.TID=paramTID}(\mathbf{T}))$
```

Fig. 4. Computing the domain input and output types of a task.

Figure 3(b) demonstrates a part of the task graph corresponding to the Purchasing process. Defining processes based on the natural flow of business objects between tasks has a number of advantages for the purpose of providing user guidance. For example, given a task, we can determine the tasks that precede it and the tasks that may follow it by using just the data from the Task and Interface modules. In comparison, data mining approaches have to rely on having significant amounts of usage data to provide a similar kind of guidance. Our approach also produces accurate descriptions of process and process instances, whereas data mining algorithms are inherently affected by noise. Furthermore, given the data in the Logging module, we can present the users with a full history of the process instance that they are working on, as shown later in this section. An illustration of one such process instance derived from empirical data that corresponds to the process in figure 3(b) is depicted in figure 3(c).

Definition 4. *We say that task a precedes task b and that task b follows task a in a given process p if $a, b \in p$ and there is a path from a to b in the task graph.*

In the multi-user environment of ERP systems, the precedence relationship between tasks does not necessarily correspond to the temporal order of the task instances involved in a process. Instead, a precedes task b reflects that a is involved in producing input to b, and, in turn, b is involved in handling the output from a.

An algorithm that computes the set of tasks preceding a given task in a specified process is depicted in figure 5. Procedure *PrecedingTasks* is a breadth-first traversal of the process subgraph of the task graph starting from the given task in the reverse direction of the arrows. The procedure starts from the given task *paramTID* (step 2), identifying all of its input types (step 6) and adding the tasks within the process that produce objects of those types to the set Θ_1 (steps 7,8). The same process is performed for each task in Θ_1 and so on until no new tasks (i.e. tasks that are not already found in the union of all visited tasks $\cup_{i=0}^{n-1}\Theta_i$) are discovered. The set of tasks following a given task is computed by a similar traversal in the direction of the arrows.

```
Procedure PrecedingTasks
Input: task id $paramTID$, process id $paramPRID$
Output: set of tasks preceding task $paramTID$ in process $paramPRID$
1  $n = 0$
2  $\Theta_0 = \{paramTID\}$
3  do
4      $n = n + 1$
5      for each $taskTID \in \Theta_{n-1}$
6          $InputTypeSet = DInput(taskTID)$
7          for each $objType \in InputTypeSet$
8              $\Theta_n = \Theta_n \cup ProducerTasks(objType, paramPRID) - \cup_{i=0}^{n-1} \Theta_i$
9  while $(\Theta_n \neq \emptyset)$
10 return: $\cup_{i=1}^{n} \Theta_i$
```

```
Procedure ProducerTasks
Input: domain object type $paramObjType$, process id $paramPRID$
Output: task ids for task from process $paramPRID$ outputting objects of type $paramObjType$
1 return: $\pi_{\mathbf{T}.TID}(\sigma_{\mathbf{T}.DOutputType=paramObjType \wedge \mathbf{PRT}.PRID=paramPRID}(\mathbf{T} * \mathbf{PRT}))$
```

Fig. 5. Computing the set of tasks preceding a given task in a specified process.

Task instance graph and process instance identification. The TIL model also provides for an easy and noiseless reconstruction of process instances, i.e. sets of task instances corresponding to a specified process, regardless of the order in which they have been executed and the number of users involved.

We introduce below two auxiliary procedures, $TIIn$ and $TIOut$, which stand for Task Instance Input and Output, respectively. Shown in figure 6, these procedures return the actual inputs and output objects of a given task instance. Both are used in a process instance identification procedure, whose definition follows in figure 7. The identification procedure is based on a domain object produced as an output and can be best understood as a breadth first traversal of the *task instance graph*.

Procedure $TIIn(paramTIID)$ returns a list of $(objectType, objectID)$ pairs for those objects entered into the entry fields associated with $paramTIID$. To determine which object id was entered, a query selects the chronologically last value of an entry field recorded in the **UserEntry** relation. To perform this selection, steps 1-2 of the procedure compute the complete set of user entries into the entry fields associated with task instance $paramTIID$. Based on the timing of the user entries, steps 3-4 produce the set of final values for each entry field, i.e. the values that were actually submitted. From those final values and the description of their domain type stored in the **InputControl** relation, step 5 composes a set of $(objectType, objectID)$ pairs, where $objectID$ is the value entered in the field, and $objectType$ specifies its domain type.

$TIOut(paramTIID)$ returns the output object type and id of the task instance $paramTIID$ from the **Task** and **TaskInstance** relations.

```
Procedure TIIn
Input: task instance id paramTIID
Output: set of pairs (objectType, objectID) used as input to paramTIID
1 $J = \sigma_{\mathbf{TI}.TIID=paramTIID}(\mathbf{TI}) * \mathbf{EF} * \mathbf{IC} * \mathbf{UE}$
2 $K = \sigma_{\mathbf{IC}.DTable!=null \wedge \mathbf{UE}.V_e!=null}(J)$
3 $L = \rho_{Last(EFID,t_e)}({}_{EFID}\mathcal{F}_{MAX_{t_e}}(\mathbf{UE}))$
4 $M = \sigma_{\mathbf{UE}.t_e=Last.t_e}(K * L)$
5 $R = \rho_{TIIn(objectType,objectID)}(\pi_{\mathbf{IC}.DTable,\mathbf{UE}.V_e}(M))$
6 return: $R$
```

```
Procedure TIOut
Input: task instance id paramTIID
Output: pair (objectType, objectID) describing the type and value of domain output
of paramTIID
1 return: $\pi_{\mathbf{T}.DTable,\mathbf{TI}.OutPKVal}(\sigma_{\mathbf{TI}.TIID=paramTIID}(\mathbf{TI} * \mathbf{T}))$
```

Fig. 6. Procedures computing the input and output of a specified task instance.

Definition 5. *A* task instance graph *is a labeled directed graph in which the set of nodes corresponds to the task instances, and a link from task instance a to task instance b with label* $o = TIOut(a)$ *is drawn whenever* $TIOut(a) \in TIIn(b)$ *and* $a.t_e < b.t_s$*, where* t_s *and* t_e *refer to the task instance start and end time.*

The process instance information can be explored in a number of useful ways, including:

1. identifying all complete or incomplete instances of a given process, or for a given user,
2. given an object, such as a goods receipt, identifying all the task instances within the process involved in creating that object, and
3. identifying all instances of a process that use a given object, such as a purchase requisition, as their input.

As an illustration, figure 7 presents a procedure called *ObjectHistory* that determines, for a given process and a given domain object, the part of that process that led to the current state of that object. The algorithm is a breadth-first traversal of the task instance graph induced by the Logging module. The traversal is performed in the reverse direction of the arrows, starting from the chronologically latest task instance that has the given domain object as its output. The traversal is complete when all task instances that are a part of the given process are identified. The output of the procedure is a set of task instances that correspond to the steps of the input process up to the point of the latest modification of the input object.

Procedure *Producer*, shown in figure 7, is called by *ObjectHistory* and returns the task instances corresponding to a given process $paramPRID$ that have a specified object as their output. Only task instances that ended before a specified time are returned.

```
Procedure ObjectHistory
Input: process $PRID$, domain table name $objType$, domain object identifier $objID$
Output: a set of task instances from TI that correspond to the process $PRID$ and precede the latest task instance outputting the object $objID$
1  $TIs = Producer(objType, objID, CurrentTime())$
2  $latestTime = \mathcal{F}_{MAX_{t_e}}(TIs)$
3  $\Theta_0 = \pi_{TIID}(\sigma_{\mathbf{TI}.TIID \in TIs \wedge \mathbf{TI}.t_e = latestTime}(\mathbf{TI}))$
4  $n = 0$
5  while $(\Theta_n \neq \emptyset)$
6      $n = n + 1$
7      $\Theta_n = \emptyset$
8      for each $ti \in \Theta_{n-1}$
9          $\Upsilon_n = TIIn(ti)$
10         for each $tiinput \in \Upsilon_n$
11             $\Theta_n = \Theta_n \cup Producer(tiinput.objectType, tiinput.objectID, ti.t_e) - \cup_{i=1}^{n-1} \Theta_i$
12 end while
13 return: $\cup_{i=1}^{n} \Theta_i$
```

```
Procedure Producer
Input: process $parPRID$, domain table name $objType$, domain object identifier $objID$, time value $t$
Output: a set of task instances from TI ending before or at $t$ of tasks from process $PRID$ that outputted object $objID$ of type $objType$
1 $K = \pi_{TIID}(\sigma_{\mathbf{PRT}.PRID = parPRID \wedge (objType, objID) \in TIOut(\mathbf{TI}.TIID) \wedge \mathbf{TI}.t_e <= t}(\mathbf{TI} * \mathbf{T} * \mathbf{PRT}))$
2 return: $K$
```

Fig. 7. Determining the object's history within a specified process.

ObjectHistory(*paramPRID*, *objType*, *objID*) starts by calling *Producer* to obtain all task instances that had the parameter object as their output. From that set, steps 1-3 determine the task instance that ended most recently (we assume a domain object cannot be edited simultaneously by different task instances). That task instance, stored in Θ_0, is the starting point of the traversal. The traversal can be characterized as a series of computations of sets Θ_n, for $n >= 1$, comprised of task instances adjacent to those in Θ_{n-1} in the task instance graph, which continues until no new task instances can be discovered. Only task instances corresponding to the tasks in the specified process $paramPRID$ are considered. The procedure returns a set of all task instances discovered via the traversal. This set contains all instances of the tasks from the given process linked via the input-output chain that have the given object as their output.

The data model and algorithms presented in this section are used in the illustrative examples presented in the next section.

5 Empirical Validation

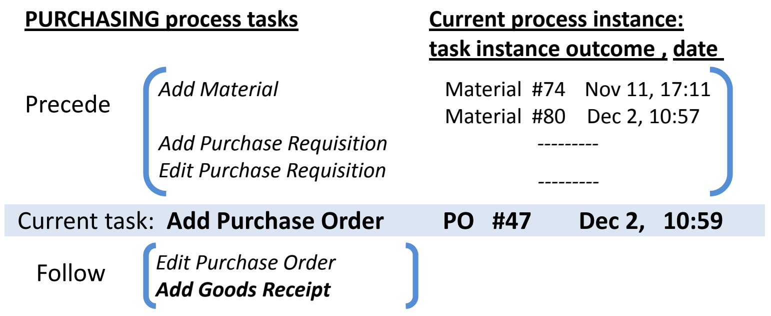
To evaluate how the TIL framework meets its design goals and evaluate its usefulness for supporting collaborative system-user interactions, we have built a prototype ERP system that utilizes the TIL model. We have conducted an emulation of ERP usage in an organization using our prototype in a laboratory setting. The emulation involved 15 users performing typical ERP tasks over a period of 27 days, with overall logged usage time of a little under 12.5 hours. There were 39 user sessions that resulted in 6,691 separate user entries. The users accessed a total of 15 different task pages, which created approximately 450 different task instances.

We have tested all algorithms presented in the previous section on the usage data collected during the emulation. Here, we present examples that demonstrate the usefulness of our framework for supporting design principles DP2 and DP4 using the emulation data.

5.1 Example 1: implementing Design Principle 2

Design principle 2 mandates that the user performing a business process be assisted by having the system display navigational guidance through completed and remaining tasks. This guidance should take into account the task and process context. It is easy to see that the TIL representation and algorithms presented in the previous section directly support derivations of context and process guidance information.

Figure 8 shows a design of an interactive display visualizing the tasks that precede and follow the user's current task. This display was constructed from the emulation data using the data and algorithms presented in the previous section. It includes the Purchasing process task information and references to the task instances related to the *Add Purchase Order* task instance.



	PURCHASING process tasks	Current process instance: task instance outcome , date
Precede	*Add Material*	Material #74 Nov 11, 17:11 Material #80 Dec 2, 10:57
	Add Purchase Requisition	---------
	Edit Purchase Requisition	---------
Current task:	**Add Purchase Order**	**PO #47 Dec 2, 10:59**
Follow	*Edit Purchase Order*	
	Add Goods Receipt	

Fig. 8. An interactive display showing a Precede/Follow list, providing quick access to related tasks, objects and task chronology.

The left side of the display in figure 8 specifies the tasks, separated into those that can precede the current task and those that can follow it. As defined in section 4.3, preceding tasks are those that are involved in producing the input to the current task and are obtained by executing the *PrecedingTasks* procedure (fig. 5), which derives the information from the **Task** and **Interface** modules of TIL. The Precede/Follow list highlights in boldface the tasks that are required for the process. It also serves as a useful reminder of the tasks related by the input-output flow and as a navigation tool: given that each task is associated with its interface page specification, a task name can also link to the appropriate task page.

The number of instantiations of each preceding task and the timing of each is included in the instance-specific information on the right side of the interactive display. The figure shows that the currently active task instance of *Add Purchase Order* was preceded by two *Add Material* instances. The other two preceding tasks are optional, and are not present in this process instance. This data is produced using the algorithm presented in figure 7, which returns a list of task instances within the process that are involved in producing the object created by the current task. The data includes the objects created by each preceding task instance and the date that the task instance was completed. Clicking on the object description (e.g. *Material #74*) will display the object. The user can also sort the list of preceding instances by the completion date to see the actual chronology of the process instance.

5.2 Example 2: Implementing Design Principle 4

A core proposition of DP4 is that the system should make use of what it knows about the user, the organization, the tasks, and the context to provide faster access to more likely choices. Software systems often do make use of prior user interactions for assisting with data entry. For example, in filling out a form on the Web, the browser will typically display one or more values that the user had previously entered to a field. With the knowledge represented by the TIL model, the system can provide access to values previously entered by the current user as well as by other users performing a particular task. The latter can be especially helpful to users with limited usage histories of their own. The granularity of usage data in our model also makes it possible to determine the users experience at a detailed level, so that appropriate assistance can be offered not only to users who are novices with the system overall, but also to those with limited experience in a particular task or even with a particular component within a task.

To demonstrate the type of information available to the system for use in tailoring the support it offers on a task-by-task basis, table 1 contains data from our laboratory study showing the users who submitted the Purchase Requisition (PR) Enter Header and Defaults page during a two-day period, the number of times they submitted, and the date of their most recent submission.

For those users with recent and frequent experience in submitting purchase requisitions, default values based on prior entries are likely to be the most useful. For someone with little or no experience, however, knowledge of the values entered by other users performing the same task can be very helpful. Values previously entered to a field can be sorted by frequency of entry, the most recent

User ID	Frequency	PR Line Items - Most Recent Submission
user10	2	24-NOV 13:15
user9	2	24-NOV 12:16
user8	2	23-NOV 14:55
user7	3	23-NOV 14:27
user6	2	23-NOV 10:50

Table 1. Count of submissions and time stamp of most recent submission of PR Header and Defaults page during a two-day period

date of entry, or any other useful property. Table 2(a) shows the values entered by all users into the Plant field in the PR header and defaults page during a two-week time period. The most frequently entered value during that time period was 15, which was also the value entered most recently. This type of information can be used in guiding a user with limited experience in filling out any form within the system.

(a)

Value	Frequency	Most Recent Entry Date
15	3	2-DEC 14:34
11	2	2-DEC 13:34
10	2	2-DEC 9:23
12	2	24-NOV 13:15
14	2	2-DEC 10:56
13	1	2-DEC 12:45
7	1	2-DEC 10:45

(b)

Input Control (IC)	IC Name	Data Type	Access Counts
1	Plant	int	67
2	Delivery Date	DATE	60
3	Storage Location	int	88
5	Vendor	int	65
601	Add Plant	menuitem	0
602	Edit Plant	menuitem	0
701	Add Vendor	menuitem	0
702	Edit Vendor	menuitem	0

Table 2. (a) User-entered values during two-week period into Plant field in PR Header and Defaults page; (b) Access counts for all input controls in the PR Header and Defaults page

DP4 also specifies that, if a choice of data or action is obvious, the system should have the option of enacting it, with the user able to replace or cancel that action. If a user almost always enters the same plant value in filling out a PR, for example, then it would make sense for the system to enter that value for the user. Similarly, if the user typically enters multiple PRs, as evidenced from the log, then the system should let the user cycle through the PR process multiple times, while also providing easy access to other frequently performed tasks.

The data captured to the usage log also provides insights into the practices of the organization that can be used for assisting the user. As an example, consider the fields that the organization requires users to fill in versus those required by the ERP system. While the latter may (or may not) be marked as required in the system, there is typically no discernible way for users to see what fields must be entered in order to adhere to organizational practices; this is because customization is costly and difficult to maintain when systems are upgraded. The system, however, does have knowledge of which fields are most often completed, or left blank; which options are most typically selected, or ignored, etc. Highlighting fields that are typically filled in can help improve the users' efficiency in filling out forms, particular those with which they have less experience.

As an example, table 2(b) shows the number of times each of the input controls in the PR header and defaults page were accessed. While storage location is not a required field, it was the most frequently accessed field. Table 2(b) also shows that none of the menu items were accessed from this page, as there were other ways of navigating that were chosen instead. Because ERP systems are designed to meet the needs of a vast array of users in varying industries, there will typically be fields or options on every page that are not needed by particular groups of users. The system can be designed to not include those components when rendering the interface. In particular, the removal of fields that are never filled in because they are not relevant to a particular organization's practices can lessen interface complexity and improve user efficiency. For other components, such as the menu items in Table 2(b) and other navigational aids, it could be that the user is just not aware of them but they could actually be beneficial (as evidenced by another group of users making frequent use of them, for example). The system could be designed to direct the user's attention to hitherto unexplored options based on its knowledge of overall system usage.

6 Conclusions and Future Work

We have presented a framework, consisting of the TIL data model and algorithms, that was designed as a foundation for implementing design principles for achieving greater ERP system usability. The framework was implemented within an ERP prototype and tested using data obtained in a laboratory emulation of ERP usage in an organization. The evaluation confirmed that the TIL model meets its design goals of supporting context-aware system interactions by enabling real-time querying of contextual and historical information in support of the design principles from section 3.

The task and process specifications contained within the TIL model structure alone (without the usage log data) make it possible to identify all of the tasks that lead to the creation of a specific type of object and all of the subsequent tasks in which that output can be used. The TIL model can also be exploited for providing support to users by informing them, for example, of the flow of outputs through the system leading to the task currently being worked on. Analysis of the usage data within the TIL model provides the larger picture of the many possible ways that users can complete processes with the system, which can be applied to navigational support, guiding the user in input choices and actions, and providing access to data and actions that are most likely to be useful.

A limitation of the presented framework is that tasks are described as having only one output: while many ERP tasks can be characterized in this way, there are some tasks that produce more than one object type. The model and algorithms can be extended to handle multiple output objects in a straightforward way.

Compared to the related work in workflow mining, which addresses some of the same aspects of system behavior, our approach is both model- and log-data-driven rather than being based solely on usage log data. This results in several advantages, including accurate and complete process instance identification regardless of the number of process instances in the log, and the ability to provide user guidance that is based on the model of tasks, processes and object flow.

In this paper, we provided examples illustrating the utility of the framework for implementing two of the four principles, DP2 and DP4. The framework also provides a foundation for several aspects of the other two principles. Indeed, the declarative description of tasks and related user interface components enables the easy customization of the system's vocabulary. The process descriptions in the TIL model provide a mechanism for process customization, as required by DP1. DP3 involves reasoning about errors, which is not addressed by the model presented here, but the TIL model already contains components that are a necessary part of error-related support of the user. In the future, we will extend the framework to fully support all four design principles. We will also develop proof-of-concept implementations of the kinds of interface features described in section 5 and evaluate them in user studies.

References

1. van der Aalst, W.M.P.: Process discovery: Capturing the invisible. IEEE Comp. Int. Mag. 5(1), 28–41 (2010)
2. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer Publishing Company, Incorporated, 1st edn. (2011)
3. van der Aalst, W.M.P., Pesic, M., Song, M.: Beyond process mining: From the past to present and future. In: CAiSE. pp. 38–52 (2010)
4. van der Aalst, W.M.P., Reijers, H.A., Weijters, A.J.M.M., van Dongen, B.F., de Medeiros, A.K.A., Song, M., Verbeek, H.M.W.E.: Business process mining: An industrial application. Inf. Syst. 32(5), 713–732 (2007)
5. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. Knowl. Data Eng. 16(9), 1128–1142 (2004)
6. Agrawal, R., Gunopulos, D., Leymann, F.: Mining process models from workflow logs. In: Proceedings of the 6th International Conference on Extending Database Technology: Advances in Database Technology. pp. 469–483. EDBT '98, Springer-Verlag, London, UK (1998), http://dl.acm.org/citation.cfm?id=645338.650397
7. Babaian, T., Lucas, W.T., Xu, J., Topi, H.: Usability through system-user collaboration. In: DESRIST. pp. 394–409. Lecture Notes in Computer Science, Springer (2010)

8. Bratman, M.E.: Shared cooperative activity. Philosophical Review 101(2), 327–341 (1992)
9. Brusilovsky, P., Cooper, D.W.: 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. pp. 23–30. ACM Press, New York, NY, USA (2002)
10. Cooprider, J., Topi, H.and Xu, J., Dias, M., Babaian, T., Lucas, W.: A collaboration model for ERP user-system interaction. In: Proceedings of the 43rd Hawaii International Conference on System Sciences (HICSS-2010) (2010)
11. Dustdar, S., Hoffmann, T., van der Aalst, W.M.P.: Mining of ad-hoc business processes with teamlog. Data Knowl. Eng. 55(2), 129–158 (2005)
12. Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems, Fourth Edition. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2003)
13. Grosz, B.G., Kraus, S.: Collaborative plans for complex group action. Artificial Intelligence 86(2), 269–357 (1996)
14. Günther, C.W., Rozinat, A., van der Aalst, W.M.P.: Activity mining by global trace segmentation. In: Business Process Management Workshops. pp. 128–139 (2009)
15. Hamerman, P.: ERP applications 2007: Innovation rekindles. Forrester Research (2007)
16. Hestermann, C.: Key issues for enterprise resource planning. Gartner (2009)
17. Iansiti, M.: ERP end-user business productivity: A field study of SAP & Microsoft: Keystone strategy. http://download.microsoft.com/download/4/2/7/427edce8-351e-4e60-83d6-28bbf2f80d0b/KeystoneERPAssessmentWhitepaper.pdf (downloaded 12/21/2009) (2007)
18. Ivory, M.Y., Hearst, M.A.: The state of the art in automating usability evaluation of user interfaces. ACM Computing Surveys 33(4), 470–516 (2001)
19. Jacko, J.A., Sears, A. (eds.): The human-computer interaction handbook: fundamentals, evolving technologies and emerging applications. L. Erlbaum Associates Inc., 2 edn. (2008)
20. Jameson, A.: Adaptive interfaces and agents. In: Sears, A., Jacko, J.A. (eds.) The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications, pp. 433–458. CRC Press, Boca Raton, FL, 2nd edn. (2008)
21. Linton, F., Joy, D., Schaefer, H.P., Charron, A.: Owl: A recommender system for organization-wide learning. Educational Technology & Society 3(1) (2000)
22. Otter, T.: Case study: Ness combines consumer application ease of use with erp robustness. Gartner (2008)
23. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. 33(1), 64–95 (2008)
24. Schonenberg, H., Weber, B., Dongen, B., Aalst, W.: Supporting flexible processes through recommendations based on history. In: Proceedings of the 6th International Conference on Business Process Management. pp. 51–66. BPM '08, Springer-Verlag, Berlin, Heidelberg (2008)
25. Terveen, L.G.: Overview of human-computer collaboration. Knowledge-Based Systems 8(2-3), 67–81 (1995)
26. Topi, H., Lucas, W., Babaian, T.: Identifying usability issues with an ERP implementation. In: Proceedings of the International Conference on Enterprise Information Systems (ICEIS-2005). pp. 128–133 (2005)