An ontology-based approach for detecting knowledge intensive tasks

Andreas S. Rath, Didier Devaurs, Stefanie N. Lindstaedt

An Ontology-Based Approach for Detecting Knowledge Intensive Tasks

Andreas S. Rath
Know-Center GmbH., Inffeldgasse 21a/II, 8010 Graz, Austria
arath@know-center.at

Didier Devaurs
CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse Cedex 4,
France and Université de Toulouse ; UPS, INSA, INP, ISAE;
UT1, UTM, LAAS ; F-31077 Toulouse Cedex 4, France
devaurs@laas.fr

Stefanie N. Lindstaedt
Know-Center GmbH., Inffeldgasse 21a/II, 8010 Graz, Austria &
Graz University of Technology, Graz, Austria
slind@know-center.at

***Abstract*— In the context detection field, an important challenge is automatically detecting the user's task, for providing contextualized and personalized user support. Several approaches have been proposed to perform task classification, all advocating the *window title* as the best discriminative feature. In this paper we present a new ontology-based task detection approach, and evaluate it against previous work. We show that knowledge intensive tasks cannot be accurately classified using only the window title. We argue that our approach allows classifying such tasks better, by providing feature combinations that can adapt to the domain and the degree of freedom in task execution.**

Categories and Subject Descriptors
H.1.2 [Models and Principles]: User/Machine Systems - *Human information processing*; I.5.2 [Pattern Recognition]: Design Methodology - *Classifier design and evaluation*

Keywords: user context detection, context-awareness, user context ontology, task detection, classification

I. Introduction

"Understanding context is vital" [1] and *"context is key"* [2] signal the growing interest in the context detection field. Context is everywhere. In the information retrieval area context is exploited for personalizing search [3], [4]. In personal information management the user context is related to projects [5], tasks [6], processes [7], topics[8] and notes [9]. Context is also used in task and process management for discovering tasks [10], [11], [12] and process flows [13]. In technology-enhanced learning, context is exploited to provide appropriate learning material [14].

Automatic user task detection is an important challenge in the area of context detection [2], [15]: if her current task is detected automatically the user can be supported with relevant information (such as learning and work resources) or task guidance. A classical approach is to model task detection as a machine learning problem, specifically a classification problem: while the user is performing an unknown task on her computer desktop, some contextual data is captured by software sensors. Based on this captured sensor data a representation of the task is constructed in form of different types of features. These features are used as input to classification algorithms that provide as output a label for the task.

Several approaches have been proposed, that successfully perform task detection from recorded user contextual data [16], [17], [18], [19], [11], [12]. Besides, it has been shown that, among the recorded data, the *window title* feature presents a very good discriminative power for task classification [17], [19], [11], [20]. If this was generally confirmed, this would imply that there is no need in recording sophisticated contextual data to perform task detection: keeping track of the *window title* alone would be sufficient. However, we identified that the success of this feature was due to the fact that the inherent goals of the tasks investigated in other research work were associated with a specific resource or with an application that could be identified in the content of the *window title*.

In this paper we present three main contributions in the area of user context detection and task detection. First we present our user interaction context ontology (*UICO*) for user context modeling, capturing and analysis in Section III. We developed our UICO following a bottom-up approach, starting from the user interaction context data provided by the software sensors we implemented [21].

As a second contribution we introduce our ontology-based task detection approach (*UICO approach*) which is based on our previous work [17]. The ontology-based conceptualization of the user interaction context enabled us to derive a set of 50 features covering all aspects of the available contextual data. The features engineered from the UICO are used to create task instances for classification purposes. This approach is different from our previous work which only used text-based features for task detection without an underlying

ontology. We do not necessarily promote the use of all 50 features together, as this could prove computationally costly. Rather, we see our approach as being adaptable, in the sense that different small-sized feature combinations could provide good classification accuracy, depending on the domain in which task detection is performed.

As a third contribution we show that, when considering tasks involving a great amount of freedom in their execution or in the produced result, the *window title* alone does not achieve good accuracy results during task classification, and that more sophisticated feature combinations are needed to achieve a high task detection accuracy.

The paper is organized as follows: first, we present the related work and the investigated datasets in the task detection area. Section III gives the details of our UICO task detection approach. In Section IV, we describe the methodology we followed to analyze our task detection results. In Section V our first laboratory experiment including the resulting dataset is described for evaluating our UICO approach on tasks similar to those already investigated in task detection research in terms of achieved accuracy. In Section VI we elaborate on our second laboratory experiment and the performed evaluations. Furthermore we discuss the limits of the *window title* feature and present a feature combination achieving better results. Finally, we draw some conclusions and present our future work.

II. Related Work & Datasets

By task detection we mean *task class detection* also referred to as *task classification*, as opposed to *task switch detection*. Task switch detection involves predicting when the user switches from one task to another [11], [12]. Task classification deals with the challenge of classifying usage data from user task execution into task classes or task types. Automatic task detection is classically modeled as a machine learning problem, and more precisely a classification problem. This method is used to recognize Web based tasks [18], tasks within emails [16], [20] or tasks from the complete user's computer desktop [17], [19], [11], [20], [12].

Usually, solving this classification problem is based on the following steps: (i) The user context data is captured by system and application sensors. (ii) Features, i.e. parts of this data, are chosen to build classification training instances, which is done at the task level. (iii) To obtain valid inputs for machine learning algorithms, these features are first transformed into attributes [22]. This transformation may include data preprocessing operations, such as removing stopwords [17], [19] and application specific terms [11], or constructing word vectors. (iv) Attribute selection [17], [20] (optional step) is performed to select the best discriminative attributes. (v) Finally, classification/learning algorithms are trained and tested.

In SWISH [11] about four hours of real usage data observed from a single user was recorded, which gave five different tasks (one instance per class) such as "Harry Potter book" or "Expedia flight trip" for example. In Dyonipos [17] three datasets were studied (Dyo1, Dyo2, Dyo3) as shown in Table I. Dyo1 contains 218 task instances collected from 14 participants, and distributed among the following task classes: "filling the official journey form", "filling the official cost recompense form for the official journey", "creating an application for leave", "planning an official journey" and "organizing a project meeting". Dyo2 and Dyo3 contain the same 140 task instances, collected from one user, but are based on different task classes. In Dyo2 the classes are: "email reading/writing", "paper writing", "search", "documentation writing" and "collecting information". In Dyo3 the classes are: "email communication", "organizational/administrative task", "writing a document" and "read". The two datasets used for evaluating TaskPredictor 1 [20] contain 96 and 81 tasks respectively (one instance per class), performed by two users. In APOSDLE [19] a single user recorded several instances of the following task classes: "market analysis", "product design and specification", "find and contact suppliers", "contract placement" and "triggering production".

Data	f	l	t	g	a	p	r
Dyo1	A C W	NB	5	300	83.5%	0.95	0.85
Dyo2	W	KNN	5	156	73.6%	0.91	0.75
Dyo3	W	KNN	4	188	76.4%	0.90	0.74
Swish	W	PLSI	5	-	70%	0.49	0.72
TP1a	W P U	N+S	96	200	-	0.8	-
TP1b	W P U	N+S	81	200	-	0.8	-
Apo	C P W	SVM	5	200	85%	-	-

TABLE I

Overview of the parameters and performance achieved by previous approaches. Data: dataset used for the evaluation (Dyo1, Dyo2 and Dyo3 come from Dyonipos [17]; Swish can be found in [11]; TP1a and TP1b come from TaskPredictor1 [20]; Apo comes from APOSDLE [19]). f: used features (A: Application, C: Content of document, W: Window title, P: File path, U: Web page URL). l: used learner/classifier (NB: Naïve Bayes, KNN: k-Nearest Neighbor, PLSI: special version of Latent Semantic Indexing, SVM: Linear Support Vector Machine, N+S: combination of NB and SVM). t: number of task classes. g: number of attributes. a: accuracy. p: micro precision. r: micro recall.

Table I presents an overview of the classification configuration leading to the best results achieved by those approaches. All of them involve the *window title* feature, which proved to be very efficient based on the obtained accuracy and/or precision. It shows that no classifier clearly stands up as an obvious best choice. In terms of number of attributes used for building the training instances, it seems that about 200-300 are sufficient to achieve good results.

III. User Interaction Context

A. Recording and Modeling the User Context

Our view of the "*user context*" goes along with Dey's definition that context is *"any information that can be used*

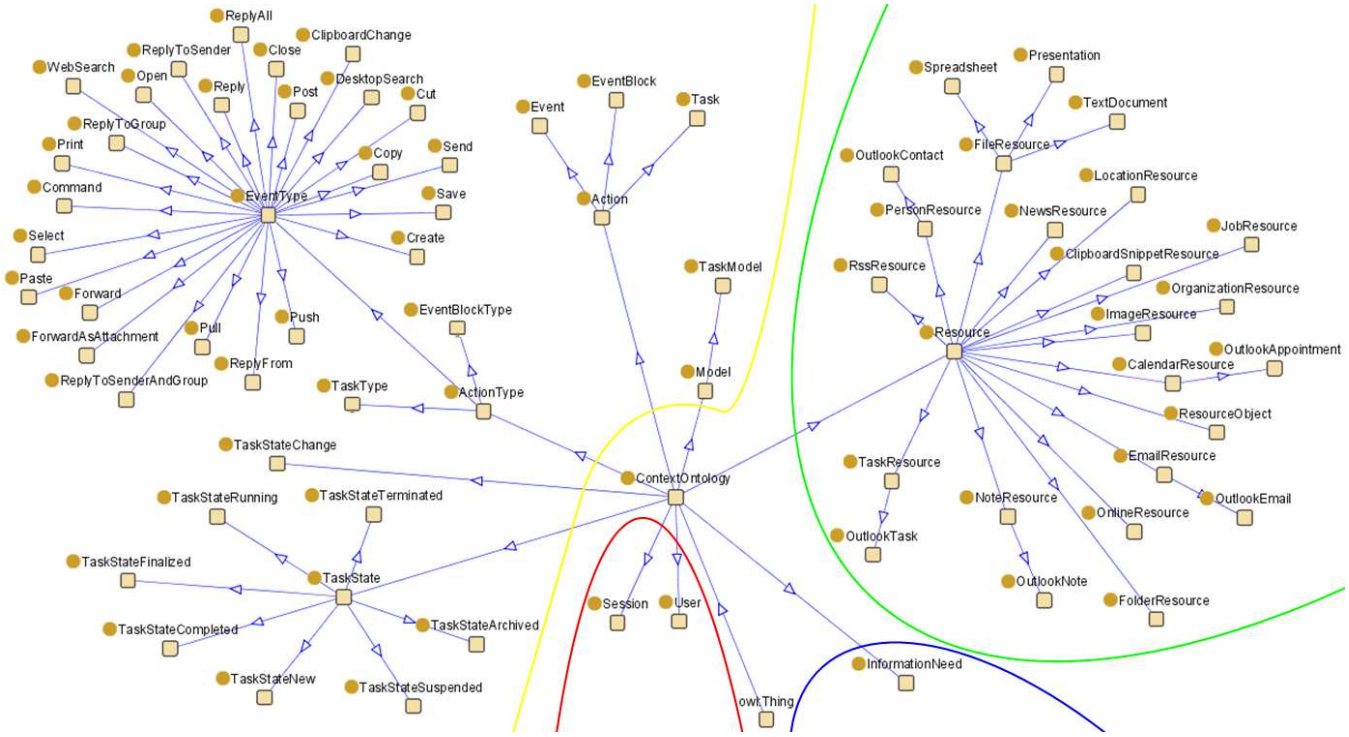

Fig. 1. The concepts of the user interaction context ontology (UICO) visualized in the Protégé tool. This figure shows in the left area the *action dimension*, in the right area the *resource dimension*, and in the bottom area the *user dimension*.

to characterize the situation of entities that are considered relevant to the interaction between a user and an application, including the user and the application themselves" [15]. We refine Dey's perspective by focusing on the *user interaction context* that we define as "*all interactions of the user with resources, applications and the operating system on the computer desktop"*.

The conceptual representation we propose for the user interaction context is the *semantic pyramid* [17]. At the bottom of the pyramid are events that result from the user's interactions with the computer desktop. Above are event-blocks, which are sequences of events that belong logically together, each event-block connecting the user's actions associated with a specific resource acted upon. At the top are tasks, which are grouping of event-blocks representing well-defined steps of a process, that cannot be divided into sub-tasks, and in which only one person is involved. The layers of the semantic pyramid represent the different aggregation levels of the user's actions.

Context observation mechanisms are used to capture the behavior of the user while working on her computer desktop. Low-level operating system and application events initiated by the user are recorded by context observers. This is similar to the approach followed in the contextual attention metadata area [23] and in context observation research in general [10], [19], [11], [12]. We have implemented multiple context sensors for the Microsoft Windows operating system and standard applications used by knowledge workers. For more details the interested reader is referred to [21].

B. User Interaction Context Ontology (UICO)

A context model is needed for storing the user context data in a machine processable form. Various context model approaches have been proposed, such as key-value models, markup scheme models, graphical models, object oriented models, logic-based models, or ontology-based models [24]. However, the ontology-based approach has been advocated as being the most promising one [25], [24] mainly because of its dynamicity, expressiveness and extensibility.

We have defined a User Interaction Context Ontology (UICO) containing 107 concepts (classes) and 281 properties, divided into 224 datatype properties and 57 objecttype properties. The UICO is modeled following a bottom up approach: its concepts and properties represent the user interaction context derived from the data captured by the context sensors and from the results of the algorithms analyzing this data. Our ontology is modeled in OWL-DL[1], using the Protégé tool[2]. The *ontology web language* (OWL) is a W3C standard widely accepted in the Semantic Web community for modeling ontologies. A visualization of the UICO concept hierarchy (sub-class relation) in Protégé is given in Figure 1. Four dimensions can be identified in the UICO:

- The **action dimension** consists of concepts representing user actions, task states and models:
  - The `Action` concept is refined by the sub-concepts `Event`, `EventBlock` and `Task`. Ex-

[1]http://www.w3.org/2004/OWL/
[2]http://protege.stanford.edu

amples of user Event sub-concepts are Print, Close, Save, Copy, Paste, Cut, Post, Reply, Forward and WebSearch.
  - The different types of task states are borrowed from the *Nepomuk Task Management Model* [26]. In this model, a task can be New, Running, Suspended, Completed, Terminated, Finalized or Archived.
  - The only model available at the moment is the TaskModel which is used to categorize a task.
- The **resource dimension** contains concepts used for representing resources available on the computer desktop. Examples of possible Resource are File, TextDocument, Presentation, Spreadsheet, OnlineResource, E-Mail, Folder, Organization, Location, and Person. Relations can be defined between concepts of the resource dimension and of the action dimension for modeling on which resources what kind of user actions are executed, via the objecttype property isActionOn.
- The **application dimension** is a "hidden" dimension in the sense that it is not modeled as concepts in the UICO. However, each user interaction happens within the focus of a certain application, and thus the Event concept holds information about the user interaction with an application through the datatype properties hasApplicationName and hasProcessId. Standard applications that run on the Microsoft Windows desktop usually consist of graphical user interface (GUI) elements. Console applications also have GUI elements such as the window itself, scroll bar(s) and buttons for minimizing, maximizing and closing the application. Most of the GUI elements have an associated *accessibility object* which can be accessed by context sensors. Datatype properties of the Event concept hold information about the interactions with GUI elements. In the sequel we show that these accessibility objects play an important role in automatic task detection. A resource being normally accessed and manipulated by the user within an application, there is a relation between the resource dimension and the application dimension, which is indirectly captured through the relations the action dimension has with both the resource and application dimensions.
- The **user dimension** contains only the User and Session concepts. It is related to the action dimension in the sense that each Action is associated with a User via the objecttype relation hasUser. The Session concept represents session and user login information.

C. Automatic Population of the Context Ontology

The contextual information sent by the context sensors is used as a basis for populating the context ontology, i.e. instantiating its concepts. The whole user interaction context detection pipeline is visualized in Figure 2. The Event concept can be directly instantiated by the sensor data. In order to instantiate the EventBlock concept, events have first to be aggregated, using application-specific as well as generic static rules and heuristics. A Task concept is related to all concepts instantiated during the execution of a task, and can be labelled once the task has been classified.

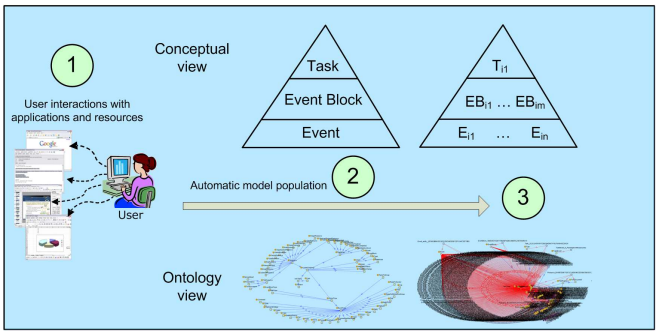

Fig. 2. Overview of the user interaction context detection pipeline, from (1) the context sensors that capture the user interaction context, to (3) the populated user interaction context model, via (2) automatic model population mechanisms (event creation and aggregation, resource discovery, etc.).

As an event-block represents a sequence of events associated with the same resource, its creation heavily relies on the resource discovery process. We use three different techniques for discovering resources. (i) The *regular expression* approach identifies resources in the sensor data based on specific character sequences predefined as regular expressions. This is used to identify files, folders, web links and email addresses for example. (ii) The *information extraction* approach extracts person, location and organization entities in text-based elements of the sensor data, using the KnowMiner framework [27]. (iii) The *direct resource identification* approach finds data about a potential resource directly in the sensor data, and build the resource by directly mapping certain fields of the sensor data to properties of the Resource concept.

These three techniques produce what we call *used resources*, in the sense that the user has interacted with them. We are also interested in unveiling relations between these used resources, and with other resources. We say that a resource is an *included resource* if its content is part of the content of another resource. A resource is a *referenced resource* if it is mentioned and identified in the content of another resource (e.g. names of persons, locations and organizations, paths of folders and files, URLs of web pages and email addresses).

D. Task Classification

Performing task detection classically consists in training classification/learning algorithms on classes corresponding to task models. This means that each training instance presented to the machine learning algorithms represents a task that has to be "labelled". Thus, training instances have to be built from features and feature combinations derived from the user context data at the task level. In the case of our ontology-based approach, this means deriving features from the data

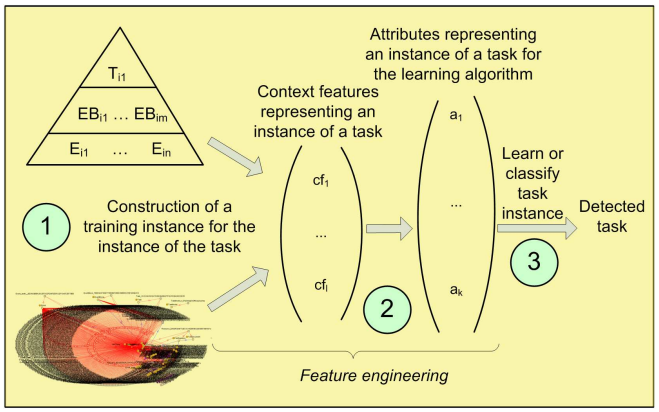

Fig. 3. Overview of the task detection pipeline: (1) the models populated with user context data, namely the semantic pyramid populated with a task (T) and its associated event-blocks (EB) and events (E), and below the corresponding part of the populated UICO, namely a `Task` concept and its associated concepts (actions, resources, etc.) as well as the relations between them; (2) the construction of features representing a task instance and their transformation into attributes (referred to as *feature engineering*); (3) the classification of the task instance.

associated with a `Task` concept. An overview of the task detection pipeline is shown in Figure 3.

Based on our UICO, we have engineered 50 features for constructing the training instances (see Figure 4). They are grouped in six categories: (i) *action*, (ii) *application*, (iii) *content*, (iv) *ontology structure*, (v) *resource* and (vi) *switching sequences*. The *action category* represents the user interactions and contains features about the interactions with applications [17], resources types, resources, key input types (navigational keys, letters, numbers), the number of events and event blocks, the duration of the event blocks, and the time intervals between event blocks. The *application category* contains the classical *window title* feature [17], [19], [11], [12], the application name feature [17] and the newly introduced accessibility objects features. The *content category* consists of features representing the content of task-related resources, the content in focus and the text input of the user. The *ontology structure category* contains features representing the number of instances of concepts and the number of datatype and objecttype relations used per task. The *resource category* includes the complete contents and URIs [20] of the used, referenced and included resources, as well as a feature that combines all the metadata about the used resources in a 'bag of words'. The *switching sequence category* comprises features about switches between applications, resources, event types and resource types.

We use the machine learning toolkit Weka [22] for parts of the feature engineering and classification processes. The following steps are performed to preprocess the content of text-based features (in this sequence): (i) remove end of line characters, (ii) remove markups, e.g. `\&lg` and `![CDATA`, (iii) remove all characters but letters, (iv) remove German and English stopwords, (v) remove words shorter than three characters. We transform text-based features into attributes with the `StringToWordVector` function of Weka, and for numeric features we apply the Weka `PKIDiscretize` filter.

IV. Validating the UICO Approach

For validating our UICO approach we designed two independent experiments. The first one involved tasks similar to those studied in previous datasets, as elaborated in Section V. By similar we mean that, the objectives of the tasks were also dependent on applications and resources. For the second experiment described in Section VI we removed this dependency by designing tasks with a great amount of freedom. In both experiments we asked computer science students from Graz University of Technology to perform the tasks. They allowed the observation of their user interaction context during their task executions and made it freely available for the evaluations described in this paper. We considered these students as being experts of the investigated domain: *"tasks of computer science students at Graz University of Technology"*

Four laboratory notebooks were prepared for the experiments. We installed the operating system Microsoft Windows (XP or Vista), and the following software packages commonly used by computer science students during their curriculum: Microsoft Office, Internet Explorer, integrated development environments (Eclipse 3.x, MS Visual Studio 2008, NetBeans 6.x), editors (Emacs, JEdit++, Notepad++, Vim, Microsoft Notepad) compilers (C, C++, C#, Java, Python, Perl, Ruby) and our user interaction context observation prototype [3] [28].

Methodology for Task Detection Analysis

In our task detection experiments, we use classic machine learning algorithms from text classification. Besides, we assess their performance with the same evaluation methodology, to ensure results comparability. The following parameters are varied in order to evaluate their influence on the task detection performance: (i) the learning algorithm, (ii) the set of used features and (iii) the number of attributes generated from the features. Furthermore, the set of used features is varied by including (i) each feature individually, (ii) each feature category individually, (iii) all feature categories or (iv) the *Top* k best performing single features, with $k \in \{2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20\}$.

The evaluated learning algorithms are: Naïve Bayes (NB), Linear Support Vector Machine (SVM) with cost parameter $c \in \{2^{-5}, 2^{-3}, 2^{-1}, 2^{0}, 2^{1}, 2^{3}, 2^{5}, 2^{8}, 2^{10}\}$, J48 decision tree (J48), and k-Nearest Neighbor (KNN) with $k \in \{1, 5, 10, 35\}$. The different values for the number of neighbors k are introduced in order to explore the task detection performance with different decision boundaries. The values of the cost parameter for the SVM are borrowed from the libSVM practical guide[4]. The Weka machine learning library [22] and the Weka integration of the libSVM[5] provide the necessary tool set to evaluate these algorithms.

[3] A demo storyboard of the prototype is available at `http://purl.oclc.org/NET/knowse/ectel2009demo`

[4] http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[5] http://www.cs.iastate.edu/~yasser/wlsvm/

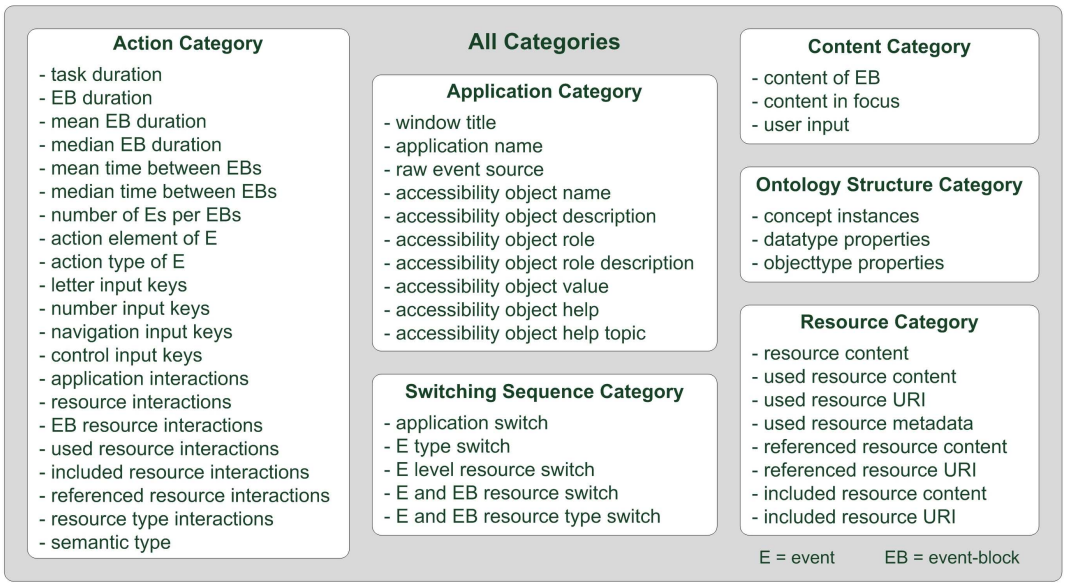

Fig. 4. Overview of all features (and their respective feature categories) engineered from our User Interaction Context Ontology (UICO).

For each learning algorithm l, each feature category ϕ and each feature f, the g attributes having the highest *Information Gain* value (IG) are selected. Information gain attribute selection is used because (i) it is one of the fastest and most popular algorithms in the text classification field, and (ii) "pre-evaluations" with more advanced attribute selection methods showed little improvement. As values for g, 50 different measure points are used. Half of them are equally distributed over the available number of attributes with an upper bound of 5000. The other half is defined by $G = \{3, 5, 10, 25, 50, 75, 100, 125, 150, 175, 200, 250, 300, 500, 750, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 5000, 7500, 10000\}$, the upper bound being 10000 or the maximum number of attributes available, whichever is less. The choice of these specific measuring points is motivated by two reasons. First, previous evaluations of task detection performance reported that good results were achieved with a low number of attributes (about 200-300) [17], [19], [20], and hence we put a special focus on similar numbers of attributes. Second, we also want to investigate the influence of using higher numbers of attributes on task classification.

The method used for evaluating the learning algorithms is stratified 10-fold cross-validation. We computed statistical values for each fold as well as the mean and standard deviation of each value across the folds. The training set and test set instances are strictly parted (i.e. constructed and preprocessed independently) to avoid any bias. Each learning algorithm is then trained on the training set and evaluated on the test set for each fold. For each task we build a training/test instance.

Comparing our UICO approach to the TaskPredictor 1, SWISH and Dyonipos approaches cannot be done directly, for the construction of the training instances differs in term of granularity. In SWISH *window switching algorithms* determine the boundaries of a training instance. In TaskPredictor *Window-Document Segments (WDSs)* are built to make a prediction. In Dyonipos one training instance per event block is constructed. In our approach, we construct one training instance for each task. Since the number of training instances per class has an influence on the classification accuracy, we focus the comparison of the different approaches on the feature engineering part. We borrow the context features used by the mentioned approaches, and apply the same feature preprocessing techniques as published in [11] for SWISH, [20] for TaskPredictor and [17] for Dyonipos.

V. Experiment 1: Comparing the UICO Approach with Previous Work

A. Experiment Design

The goal of our first laboratory experiment was to analyze the capabilities of automatic task detection for computer science tasks in a controlled setting with tasks similar to those already studied in related datasets (see Section II). We first interviewed computer science students in order to gather typical tasks they perform during their university curriculum. Then, during our first workshop, four selected computer science students discussed these tasks and chose eight of them as being the most representative. During our second workshop, the same students performed the tasks, to estimate their duration. We discarded the longest knowledge intensive task in order to not exceed 90 minutes per recording session . Thus seven task classes were studied: four were considered as **routine tasks** (Task 1: *Register for an examination*, Task 2: *Finding course dates*, Task 3: *Reserve a book in the university library*, Task 4: *Course registration*) and three as **knowledge intensive tasks** (Task 5: *Algorithm programming*, Task 6: *Prepare a scientific talk*, Task 7: *Plan a study trip*). Ten computer science students participated in the experiment. Each subject executed each task twice (in a different order), but six task instances were lost, due to a software failure.

The final dataset thus contains 134 tasks (see Table II for the exact distribution of the task instances).

#	Task Class	Task Instances
1	Register for an examination	19
2	Finding course dates	20
3	Reserve a book in the university library	20
4	Course registration	17
5	Algorithm programming	20
6	Prepare a scientific talk	19
7	Plan a study trip	19
Dataset		134

TABLE II
TASK INSTANCE DISTRIBUTION IN EXPERIMENT 1.

B. Evaluating Task Detection Performance

Table III presents the task detection performance results of the UICO and the related approaches. The best accuracy values were approximately between 90% and 95% for the various feature categories, single best features, top k feature combinations, as well as for the related approaches. These results confirm that task detection can be performed with high accuracy via machine learning algorithms, and that the *window title* has a high discriminative power (as shown previously by the SWISH, Dyonipos and TaskPredictor 1 approaches). Using this feature alone, it was possible to achieve an accuracy of 88.13%. We can also confirm that a small portion of the whole attribute set associated with a feature combination is sufficient for obtaining a good classification. For example, considering the top six features, using 100 attributes (compared to the 7970 attributes available) resulted in an accuracy of 94%. More generally, the range of 200-300 attributes suggested by the Dyonipos and TaskPredictor 1 approaches (see Table I) seems a good choice.

We also evaluated the task detection performance resulting from classifying routine task instances (among four task classes) and knowledge intensive task instances (among three task classes) independently, by splitting the dataset. The accuracy achieved was 94.64% for routine tasks and 100% for knowledge intensive tasks. However this last result is not well supported, for two reasons: (i) this was only a three-class classification problem, and (ii) the task classes involved were too different, such that it was too easy for the classifiers to find distinguishing attributes to train on. Even though the accuracy value itself is controversial, this result shows that knowledge intensive tasks could be detected as well as routine tasks.

A deeper analysis of the attributes involved in the classification models unveil three interesting peculiarities. First, the most discriminative attributes originating from the *window title* feature belong to the university information system that was used to perform Tasks 1-4: these attributes translates in English into "registration", "course" and "library". Second, regarding Tasks 5-7, the *application name* feature discriminates easily between them, since each task involves a different kind of application: an integrated development environment or the command line for Task 5, Microsoft PowerPoint for Task 6, and Microsoft Internet Explorer for Task 7. Third, five of the top ten features are related to the resources involved during task executions, and the Resource Category (encompassing all resource-related features) shows an accuracy of 87.91%. This suggests that resources play an important role in classifying tasks, which was already recognized by the Task Predictor 1 approach in its use of the *file path* feature.

Since specific applications and resources seem to play a major role in classifying tasks, we wanted to study whether this was also the case for tasks involving a high degree of freedom, i.e. tasks that do not rely on the use of certain applications or specific resources to be completed.

VI. EXPERIMENT 2: DETECTING KNOWLEDGE INTENSIVE TASKS

A. Experiment Design

Our second laboratory experiment was performed similarly to the first one (see Section V), in the same domain, but focused on investigating automatic task detection applied to usage data observed during the execution of **knowledge intensive tasks** only. The *CommonKADS* knowledge intensive task classification [29] was used to define the task classes. Eight task classes were chosen, instantiating eight categories of the *CommonKADS* classification: *Classification*: "Classify a list of computer science terms to hardware and software", *Diagnosis*: "Find the origin of a malfunction in a computer program", *Assessment*: "Assess whether a student has to pay tuition fees based on her/his application", *Prediction*: "Predict the questions of an exam based on historical data", *Design*: "Create a simple conceptual design for the software of an elevator system", *Assignment*: "Assign students to study groups lead by study assistants", *Planning*: "Plan a software project for the development of a document management system", *Scheduling*: "Create a schedule for the design and development of an electronic library book lending system".

Eighteen computer science students participated in the experiment. Before performing the tasks, they had to confirm they understood the task instructions. As they had to work on a laboratory computer, they were given ten minutes to get familiar with its capabilities and the installed programs. They were especially instructed to use the computer as much as possible, and not to use any other resource, such as pen and paper. Each subject executed each task once (*within subjects* experiment design) in a specific order generated randomly, but some of them forgot to perform a task. Besides, due to an operating system failure, some of the tasks were lost. The final dataset contains 132 tasks (see Table IV for the distribution of the task instances).

B. Evaluating Task Detection Performance

Table V presents the task detection performance results of the UICO and the related approaches. The highest accuracy (86.43%) was achieved by combining the Top 6 features. The Top 6 features were the following: (i) *accessibility*

	R_S	f	l	g	a	p	r	R_G
Feature Categories	1	All Categories	NB	2500	94.84	0.99	0.95	2
	2	Application Category	NB	500	93.30	0.99	0.94	6
	3	Resource Category	J48	200	87.91	0.97	0.87	12
	4	Ontology Structure Category	J48	359	81.59	0.96	0.82	14
	5	Action Category	NB	250	80.49	0.96	0.81	15
	6	Switching Sequence Category	NB	100	73.85	0.94	0.73	21
	7	Content Category	J48	75	50.88	0.85	0.49	23
Single Features	1	used resource metadata	J48	300	90.33	0.98	0.91	8
	2	accessibility object name	NB	50	88.79	0.98	0.89	10
	3	window title	J48	75	88.13	0.98	0.89	11
	4	datatype properties	J48	221	82.14	0.96	0.83	13
	5	resource type interactions	J48	27	79.07	0.95	0.79	16
	6	accessibility object value	KNN	75	78.35	0.95	0.79	17
	7	resource interactions	NB	50	76.15	0.95	0.75	18
	8	used resource interactions	NB	50	75.49	0.94	0.76	19
	9	used resource URI	NB	125	74.73	0.94	0.74	20
	10	application interactions	J48	65	72.31	0.93	0.71	22
Top k Features	1	Top 6	J48	100	94.07	0.99	0.94	3
	2	Top 15	NB	4000	94.01	0.99	0.94	4
	3	Top 5	NB	175	93.35	0.99	0.94	5
Dyonipos	1	content of document + window title	NB	125	95.49	0.99	0.96	1
TaskPredictor 1	2	file path + web page URL + window title	NB	300	93.24	0.99	0.92	7
Swish	3	window title	J48	50	88.90	0.98	0.89	9

TABLE III

OVERVIEW OF THE BEST TASK DETECTION RESULTS FOR THE FIRST LABORATORY EXPERIMENT (7 TASK CLASSES), EVALUATED BY STRATIFIED 10-FOLD CROSS-VALIDATION FOR EACH FEATURE CATEGORY, FOR EACH SINGLE FEATURE, FOR THE k TOP PERFORMING FEATURES COMBINED, AND FOR THE RELATED APPROACHES. THE LEARNING ALGORITHM (l), THE NUMBER OF ATTRIBUTES (g), THE ACCURACY (a), THE MICRO PRECISION (p), THE MICRO RECALL (r), THE RANKING IN THE CORRESPONDING SECTION (R_S) AND ACROSS SECTIONS (R_G) ARE ALSO GIVEN.

#	Task Class	Task Instances
1	Classification	17
2	Diagnosis	15
3	Assessment	19
4	Prediction	16
5	Design	18
6	Assignment	16
7	Planning	16
8	Scheduling	15
Dataset		132

TABLE IV

TASK INSTANCE DISTRIBUTION IN EXPERIMENT 2.

object name, (ii) *window title*, (iii) *used resource metadata*, (iv) *accessibility object value*, (v) *application interactions* and (vi) *concept instances*. This was about 20% better than the best results obtained with the Dyonipos and the Task Predictor 1 approaches, and about 30% better than with the SWISH approach that makes use of the *window title* alone. It is important to note that, although the tasks had a high degree of freedom and could be completed with different kind of applications and resources, high accuracy levels in detecting them were still possible. This confirms the choice of utilizing machine learning algorithms for the "task detection" problem.

C. Features Discriminating knowledge intensive Tasks

Even though the *window title* was the second best performing feature (cf. Table V) its performance in this experiment was much lower than in the first one (cf. Table III). While the precision was almost as high as in the first experiment, the accuracy value was about 25% lower, and the recall value was about 0.25 lower, in the second experiment. The low recall value indicates that only a small number of the task instances belonging to a given class were correctly classified. As a result, the Dyonipos, TaskPredictor 1 and SWISH approaches, that strongly focus on the *window title*, did not perform as well in this experiment as in the first one. The lower performance of the *window title* suggests that it may not present a good discriminative power while analyzing tasks involving a high degree of freedom.

In comparison to the *window title* feature, the *accessibility object name* feature (which represents the number of interactions of the user with the names of graphical user interface elements of an application) shows a higher accuracy, a higher precision and a higher recall. It seems from the results of both experiments that the *accessibility object name* is highly discriminative, for tasks with or without a high degree of freedom.

The resource-specific features (*resource type interactions*, *resource interactions* and *used resource interactions*) that

	R_S	f	l	g	a	p	r	R_G
Feature Categories	1	All Categories	J48	10000	85.00	0.97	0.86	4
	2	Application Category	J48	500	80.38	0.96	0.81	5
	3	Resource Category	J48	1500	62.86	0.92	0.64	10
	4	Action Category	NB	250	60.71	0.90	0.60	12
	5	Content Category	J48	250	59.78	0.91	0.60	14
	6	Switching Sequence Category	NB	1173	54.51	0.88	0.52	17
	7	Ontology Structure Category	KNN	75	53.02	0.88	0.54	18
Single Features	1	accessibility object name	J48	175	80.27	0.96	0.82	6
	2	window title	J48	250	63.57	0.91	0.64	9
	3	used resource metadata	J48	2000	61.43	0.91	0.60	11
	4	accessibility object value	J48	100	60.55	0.92	0.62	13
	5	application interactions	J48	50	54.67	0.89	0.58	16
	6	concept instances	KNN	10	52.09	0.86	0.51	19
	7	content in focus	J48	150	48.52	0.85	0.49	20
	8	content of EB	NB	712	47.91	0.86	0.51	21
	9	datatype properties	J48	221	47.69	0.85	0.48	22
	10	used resource URI	NB	150	46.21	0.83	0.45	23
Top k Features	1	Top 6	J48	1500	86.43	0.98	0.86	1
	2	Top 20	J48	10000	85.66	0.97	0.84	2
	3	Top 7	J48	2000	85.49	0.98	0.86	3
Dyonipos	1	content of document + window title	SVM	500	66.04	0.93	0.66	7
TaskPredictor 1	2	file path + web page URL + window title	J48	150	63.85	0.92	0.64	8
Swish	3	window title	J48	414	56.76	0.91	0.61	15

TABLE V

OVERVIEW OF THE BEST TASK DETECTION RESULTS FOR THE SECOND LABORATORY EXPERIMENT (8 TASK CLASSES), EVALUATED BY STRATIFIED 10-FOLD CROSS-VALIDATION FOR EACH FEATURE CATEGORY, FOR EACH SINGLE FEATURE, FOR THE k TOP PERFORMING FEATURES COMBINED, AND FOR THE RELATED APPROACHES. THE LEARNING ALGORITHM (l), THE NUMBER OF ATTRIBUTES (g), THE ACCURACY (a), THE MICRO PRECISION (p), THE MICRO RECALL (r), THE RANKING IN THE CORRESPONDING SECTION (R_S) AND ACROSS SECTIONS (R_G) ARE ALSO GIVEN.

are part of the Top 10 features of Experiment 1, are not among the Top 10 features of Experiment 2. This suggests that taking away the resource dependency of the tasks, i.e. allowing more freedom in the choice of the resources used during task execution, results in a lower discriminative power of resource-specific features. This is confirmed by the fact that the *used resource metadata* feature (which is built based on the metadata about the resources the user has interacted with) shows a very good performance in Experiment 1 (90.33% accuracy) but only a moderate one in Experiment 2 (61.43% accuracy). Similarly to the *window title*, its precision is high and its recall is low.

The following six features show a good performance for both experiment datasets: *accessibility object name*, *window title*, *used resource metadata*, *application interactions*, *datatype properties* and *used resource URI*.

D. Discussion

On our two datasets we analyzed the performance of four different classifiers (Naïve Bayes, Linear Support Vector Machine, J48 decision tree and k-Nearest Neighbor) with various parameter settings (see Section IV). As a result, we observe that the Naïve Bayes and the J48 decision tree algorithms perform globally better than the Linear Support Vector Machine and the k-Nearest Neighbor algorithms. More precisely, on the first dataset the Naïve Bayes algorithm performed best with the features suggested by the Dyonipos approach with an accuracy of 95.49%. The J48 decision tree algorithm was second with an accuracy of 94.07% with the "Top 6 features" combination of our approach. (cf. Table III). On the second dataset the J48 decision tree algorithm achieved the best results, far ahead of the other classifiers (cf. Table V). We have then performed a more detailed comparison of the four classifiers, on different fixed settings applied to our two datasets (cf. Table VI). This comparison confirms the superiority of Naïve Bayes and J48 decision tree.

In the first experiment, the number of attributes selected for the best runs of the classifiers was generally below 500, except for the feature combinations "All Categories" and "Top 15 features" (cf. Table III). This is quite low, especially compared to what was obtained in the second experiment, where the best settings include numbers of attributes higher than 500 (cf. Table V). A possible explanation for this increase may be the higher degree of freedom involved in the tasks performed: in order to handle this higher freedom the classifier model requires more attributes for distinguishing the tasks.

Our method for finding the best task detection setting in the context of the UICO approach involved the evaluation of all feature categories, all single features, and various combinations of the best performing features, so-called *Top k* features (with different values for k). We limited our

	Feature(s)	J48	KNN	NB	SVM
Experiment 1	UICO (All Categories)	91.15 (125)	89.62 (300)	**94.84** (2500)	54.34 (3)
	UICO (Top 6 features)	**94.07** (100)	89.62 (200)	92.64 (500)	55.93 (3)
	Dyonipos (C W)	89.56 (100)	93.96 (100)	**95.49** (125)	93.30 (50)
	SWISH (W)	**88.90** (50)	83.85 (25)	83.52 (150)	62.03 (5)
	TaskPredictor 1 (W P U)	88.96 (175)	86.65 (125)	**93.24** (300)	53.79 (3)
Experiment 2	UICO (All Categories)	**85.00** (10000)	61.32 (2500)	64.34 (10000)	51.37 (10)
	UICO (Top 6 features)	**86.43** (1500)	61.65 (2500)	59.12 (1500)	41.04 (5)
	Dyonipos (C W)	62.31 (175)	61.37 (200)	63.02 (300)	**66.04** (500)
	SWISH (W)	**56.76** (414)	51.70 (75)	49.45 (414)	36.32 (5)
	TaskPredictor 1 (W P U)	**63.85** (150)	52.97 (175)	56.10 (175)	27.25 (25)

TABLE VI

OVERVIEW OF THE ACCURACY VALUES OBTAINED BY THE DIFFERENT CLASSIFIERS (J48: J48 DECISION TREE, NB: NAÏVE BAYES, KNN: k-NEAREST NEIGHBOR, SVM: LINEAR SUPPORT VECTOR MACHINE) IN DIFFERENT SETTINGS. FOR EACH SETTING, THE BEST ACCURACY IS HIGHLIGHTED. THE NUMBER OF ATTRIBUTES IS GIVEN BETWEEN BRACKETS NEXT TO THE ACCURACY VALUE. THE ABBREVIATED FEATURES ARE: W - WINDOW TITLE, C - CONTENT OF DOCUMENT, P - FILE PATH, U - WEB PAGE URL.

evaluation to these feature combinations because of time constraints, since computing all combinations of the 50 available features is not reasonable with today's computing power. In this work, we have reduced the number of possible combinations by suggesting a set of six features. We consider finding a good combination of features for specific tasks of a given domain as fine tuning task classification, and we believe that there is not a unique feature combination that would be the best one for all settings.

VII. CONCLUSION AND FUTURE WORK

We have presented here our ontology-based user task detection approach. We have performed two large-scale laboratory experiments for evaluating its performance and for studying more thoroughly the problem of detecting knowledge intensive tasks. Compared to previous work, our approach achieved good accuracy levels. It specially confirmed that the classical *window title* feature shows good discriminative power for task classification. However, we showed that this was valid mainly for simple tasks. When more sophisticated knowledge intensive tasks are considered, the *window title* shows its limitation, and feature combinations with more features have to be considered in order to achieve good classification results. Similarly, the number of attributes used for constructing the training instances has to be increased.

In our second experiment, we isolated a combination of six features that present a good discriminative power. However, we do not claim that this combination would perform equally well on another dataset. Our message is rather that, better results are not necessarily achieved with feature combinations with a large number of features. We argue that, from the set of features we have defined, small-sized combinations showing a good discriminative power can be identified. In that sense, we see our approach as being adaptable, since it allows to tune the task detection settings based on the studied domain.

We plan to run more task detection experiments focused on knowledge intensive tasks in order to answer the important research question: "Which tasks can be automatically detected with a good accuracy?". We are interested in finding a combination of classifiers and features that achieves good results on a standard desktop computer, at a low computational cost. We also plan to study whether a classifier trained on context data recorded from a single "expert" could provide good results while applied to task instances recorded from several users. We will also investigate unsupervised learning mechanisms for identifying boundaries in the user interaction context data, in order to classify these clusters into task classes. Our aim is to develop a real-time task detection application respecting the computational power available on standard desktop computers. We envision a knowledge services framework[6] that provides a variety of intelligent and contextualized knowledge services [28], [30], such as context-aware/task-based information retrieval, user interruptibility, visual reflection, as well as personal and organizational information and task management.

Acknowledgments

The Know-Center is funded within the Austrian COMET Program - Competence Centers for Excellent Technologies - under the auspices of the Austrian Ministry of Transport, Innovation and Technology, the Austrian Ministry of Economics and Labor and by the State of Styria. COMET is managed by the Austrian Research Promotion Agency FFG.

REFERENCES

[1] S. Greenberg, "Context as a dynamic construct," *Human-Computer Interaction*, vol. 16, no. 2, pp. 257–268, 2001.

[2] J. Coutaz, J. L. Crowley, S. Dobson, and D. Garlan, "Context is key," *Communications of the ACM*, vol. 48, no. 3, pp. 49–53, 2005.

[3] J. Callan, J. Allan, C. L. A. Clarke, S. Dumais, D. A. Evans, M. Sanderson, and C. Zhai, "Meeting of the MINDS: an information retrieval research agenda," *ACM SIGIR Forum*, vol. 41, no. 2, pp. 25–34, 2007.

[6] http://en.know-center.at/forschung/knowledge_services

[4] J. C. Tang, J. Lin, J. Pierce, S. Whittaker, and C. Drews, "Recent shortcuts: using recent interactions to support shared activities," in *Proc. CHI '07*, 2007, pp. 1263–1272.

[5] W. Jones, P. Klasnja, A. Civan, and M. L. Adcock, "The personal project planner: planning to organize personal information," in *Proc. CHI '08*, 2008, pp. 681–684.

[6] T. Catarci, A. Dix, A. Katifori, G. Lepouras, and A. Poggi, "Task-centered information management," in *Proc. DELOS '07*, 2007, pp. 253–263.

[7] K. D. Fenstermacher, "Revealed processes in knowledge management," in *Proc. WM '05*, 2005, pp. 443–454.

[8] L. Sauermann, A. Bernardi, and A. Dengel, "Overview and outlook on the semantic desktop," in *Workshop on the Semantic Desktop, ISWC '05*, 2005.

[9] M. Van Kleek, M. Bernstein, D. R. Karger, and M. Schraefel, "Gui – phooey!: the case for text input," in *Proc. UIST '07*, 2007, pp. 193–202.

[10] A. N. Dragunov, T. G. Dietterich, K. Johnsrude, M. McLaughlin, L. Li, and J. L. Herlocker, "TaskTracer: a desktop environment to support multi-tasking knowledge workers," in *Proc. IUI '05*, 2005, pp. 75–82.

[11] N. Oliver, G. Smith, C. Thakkar, and A. C. Surendran, "SWISH: semantic analysis of window titles and switching history," in *Proc. IUI '06*, 2006, pp. 194–201.

[12] J. Shen, J. Irvine, X. Bao, M. Goodman, S. Kolibaba, A. Tran, F. Carl, B. Kirschner, S. Stumpf, and T. G. Dietterich, "Detecting and correcting user activity switches: algorithms and interfaces," in *Proc. IUI '09*, 2009, pp. 117–126.

[13] W. M. P. van der Aalst and A. J. M. M. Weijters, "Process mining: a research agenda," *Computers and Industry*, vol. 53, no. 3, pp. 231–244, 2004.

[14] A. Schmidt, "Bridging the gap between knowledge management and e-learning with context-aware corporate learning solutions," in *Proc. WM '05*, 2005, pp. 203–213.

[15] A. K. Dey, G. D. Abowd, and D. Salber, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," *Human Computer Interaction*, vol. 16, no. 2, pp. 97–166, 2001.

[16] M. Dredze, T. Lau, and N. Kushmerick, "Automatically classifying emails into activities," in *Proc. IUI '06*, 2006, pp. 70–77.

[17] M. Granitzer, A. S. Rath, M. Kröll, C. Seifert, D. Ipsmiller, D. Devaurs, N. Weber, and S. Lindstaedt, "Machine learning based work task classification," *Journal of Digital Information Management*, vol. 7, no. 5, pp. 306–314, 2009.

[18] A. Gutschmidt, C. H. Cap, and F. W. Nerdinger, "Paving the path to automatic user task identification," in *Workshop on Common Sense Knowledge and Goal-Oriented Interfaces, IUI '08*, 2008.

[19] R. Lokaiczyk, A. Faatz, A. Beckhaus, and M. Goertz, "Enhancing just-in-time e-learning through machine learning on desktop context sensors," in *Proc. CONTEXT '07*, 2007, pp. 330–341.

[20] J. Shen, L. Li, T. G. Dietterich, and J. L. Herlocker, "A hybrid learning system for recognizing user tasks from desktop activities and email messages," in *Proc. IUI '06*, 2006, pp. 86–92.

[21] A. S. Rath, N. Weber, M. Kröll, M. Granitzer, O. Dietzel, and S. N. Lindstaedt, "Context-aware knowledge services," in *Workshop on Personal Information Management, CHI '08*, 2008.

[22] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. San Francisco, USA: Morgan Kaufmann, 2005.

[23] M. Wolpers, J. Najjar, K. Verbert, and E. Duval, "Tracking actual usage: the attention metadata approach," *Educational Technology & Society*, vol. 10, no. 3, pp. 106–121, 2007.

[24] T. Strang and C. Linnhoff-Popien, "A context modeling survey," in *Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp '04*, 2004.

[25] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263–277, 2007.

[26] T. Groza, S. Handschuh, K. Möller, G. Grimnes, L. Sauermann, E. Minack, C. Mesnage, M. Jazayeri, G. Reif, and R. Gudjónsdóttir, "The NEPOMUK project - on the way to the social semantic desktop," in *Proc. I-Semantics '07*, 2007, pp. 201–211.

[27] W. Klieber, V. Sabol, M. Muhr, R. Kern, G. Öttl, and M. Granitzer, "Knowledge discovery using the KnowMiner framework," in *Proc. IADIS '09*, 2009.

[28] A. S. Rath, D. Devaurs, and S. N. Lindstaedt, "Contextualized knowledge services for personalized learner support," in *Proc. Demo. ECTEL '09*, 2009.

[29] G. Schreiber, H. Akkermans, A. Anjewierden, R. Dehoog, N. Shadbolt, W. Vandevelde, and B. Wielinga, *Knowledge Engineering and Management: The CommonKADS Methodology*. The MIT Press, 1999.

[30] A. S. Rath, D. Devaurs, and S. N. Lindstaedt, "Knowse: Fostering user interaction context awareness," in *Demo Proc. ECSCW '09*, 2009.