

Predicting Task Execution Times by Deriving Enhanced Cognitive Models from User Interface Development Models

Michael Quade¹
michael.quade@dai-labor.de

Marc Halbrügge²
marc.halbruegge@telekom.de

Klaus-Peter Engelbrecht²
klaus-
peter.engelbrecht@telekom.de

Sahin Albayrak¹
sahin.albayrak@dai-labor.de

Sebastian Möller²
sebastian.moeller@telekom.de

¹DAI-Labor
Technische Universität Berlin
Ernst-Reuter-Platz 7, 10587, Berlin

²Quality and Usability Lab
Technische Universität Berlin
Ernst-Reuter-Platz 7, 10587, Berlin

ABSTRACT

Adaptive user interfaces (UI) offer the opportunity to adapt to changes in the context, but this also poses the challenge of evaluating the usability of many different versions of the resulting UI. Consequently, usability evaluations tend to become very complex and time-consuming. We describe an approach that combines model-based usability evaluation with development models of adaptive UIs. In particular, we present how a cognitive user behavior model can be created automatically from UI development models and thus save time and costs when predicting task execution times. With the help of two usability studies, we show that the resulting predictions can be further improved by using information encoded in the UI development models.

Author Keywords

HCI; Model-Based Development; Automated Usability Evaluation; User Behavior Model; Simulation

ACM Classification Keywords

H.5.2 Information Interfaces and Presentation (e.g. HCI): User Interfaces

INTRODUCTION

One of the major challenges of today's user interface (UI) development is to ensure the usability of UIs when adapting to context changes. A common goal is to develop *plastic* UIs [4], which adapt to context changes while preserving certain usability properties within a predefined range. To do so, proposed properties for *plasticity*, such as observability and predictability [8], need to be expressed by a human designer at development time in a quantifiable way and require to be monitored and maintained during runtime of the application.

Another approach to achieve the goals of plastic UIs is conducting usability evaluations at development time. Usually,

the best way is carrying out tests with real users; e.g. by applying *Think Aloud* [18]. Additionally, experts can check the application's UI and interaction logic against predefined criteria; e.g. by performing a *Cognitive Walkthrough* [21] or a *Heuristic Evaluation* [18]. Carrying out any of these methods is a time consuming and expensive task. This is particularly true for UIs that adapt to users or contextual parameters because different users and context conditions need to be analyzed. Since this is impractical, applying automated usability evaluation (AUE) methods is promising [9].

AUE methods that simulate user actions are based on models of the user and the system [9]. Thus, it is possible to draw on a vast body of research dealing with *model-based evaluation* [14]. In general, model-based evaluation specifies a formal framework for applying established knowledge about how users interact with computers in a design situation. Therefore, it has similarities with expert evaluation methods. However, due to the formal nature of the framework, it is principally suitable for automation.

In this paper, we apply model-based evaluation at development time in order to address the challenges of complexity and time consumption for usability evaluations of adaptive UIs. In particular, we show how model-based evaluation is performed automatically based on runtime UI models conforming to the CAMELEON reference framework [4]. The framework structures the development process for plastic UIs during model-based UI development (MBUID). Most notably, UI development models are categorized into *task models*, modality-independent *abstract UI models* (AUI), modality-specific *concrete UI models* (CUI), and platform-specific *final UI models* (FUI). Even though most MBUID approaches differ in detail, their generic structure and development processes can be aligned to CAMELEON; e.g. approaches based on the UsiXML description language [15].

A main benefit is that by using these UI development models in a model-based evaluation approach, different stages of the prototypes and adaptation variants can be evaluated without creating system models in the required evaluation method and thus saving time and costs. Furthermore, we show that such UI development models contain valuable information about the UI which is used to render more precise evaluation results without further intervention of usability experts.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
EICS'14, June 17–20, 2014, Rome, Italy.
Copyright © 2014 ACM 978-1-4503-2725-1/14/06...\$15.00.
<http://dx.doi.org/10.1145/2607023.2607033>

The remainder of this paper is structured as follows. Below, we give an overview of related work and lead over to the description of our approach. After that, we introduce a pilot usability study and derive our findings in order to enhance our approach. We then evaluate this enhanced approach with a second usability study and discuss the results before we present a general discussion and final conclusions.

RELATED WORK

In this section, current work in the domain of MBUID is presented and we refer to how usability evaluations are introduced to the process. After that, we present benefits from model-based evaluation methods and automated tools.

Model-based UI Development and Usability Evaluations

A generic overview of potential combinations and the placement of usability evaluation methods within MBUID is presented in [1]. In general, the access to the application's task model allows calculating all potential task sequences and therefore enables using the task models for usability evaluation purposes as described e.g. in [19]. Specifically, by simulating different task performances, it becomes possible to detect potential pitfalls regarding the structure of the interaction process and to compare different variants.

The approach proposed in [7] uses MatLab to describe the system model and UsiXML to describe the UI in order to check early prototypes against a set of usability related rules and to predict human performance using a cognitive architecture. While first results have been promising, the approach does not cover an integration of UsiXML with the system model and the cognitive architecture and therefore does not cover automation for providing the evaluation models.

The TERESA authoring tool [17] supports the user-centric development of interactive applications based on UI models that are derived from task models using the ConcurTaskTree (CTT) notation [19]. TERESA allows for an early evaluation of the interaction flow by simulating the task tree and thus enables detecting possible dead ends or unreachable states.

The Multi Access Service Platform (MASP) [3] is a runtime framework for adaptive UI models. The MASP is based on executable models for multimodal UIs focusing on the domain of smart environments. MASP UIs are defined as sets of models compatible with CAMELEON. A main feature of the MASP is that the models are interconnected via mappings and can be held at runtime in order to dynamically derive the final UI; i.e. the models are executed at runtime. By this means, modifications to the models are reflected in the UI and actions on the UI lead to changes in the underlying models. During creation and testing of the task models, the MASP Task Tree Editor (MTTE) is used which is based on the CTT Editor [19]. The MTTE assists in defining the task hierarchy by structuring tasks according to their (temporal) relationships. Further, the MTTE can be used to simulate task flows by hand in order to check if all scenarios are covered.

Model-based Usability Evaluation and AUE Tools

Traditionally, model-based evaluation has been focused on execution time for routine tasks performed by skilled users.

As Kieras [14] points out, trying to explain purely cognitive tasks is complex and usually requires to understand the involved processes, whereas routine tasks mainly consist of perceptual and motor activities, which can be modeled more generally and are related to UI design. Task execution time is a popular instrumental measure to quantify efficiency as one criterion of usability. As argued by [14], the execution time of routine tasks is closely related to the number of perceptual and motor steps: a routine task involving less such steps can be performed more quickly. At the same time, a UI allowing to perform the same task with fewer steps will likely be simpler and thus more usable [14]. To encounter concerns about the models' validity, guidelines for dealing with uncertainty in the predictions have been defined, especially for the case that one of several designs is to be pursued further [14].

The GOMS approach for analytical modeling uses models of *goals*, *operators*, *methods*, and *selection* rules [5]. In a GOMS model, *goals* are used to represent the direction of interaction. In order to model how a human user would achieve these goals, *operators* are being performed on a perceptual, cognitive or motor-act level. Execution times are bound to these operators in order to predict overall task execution times for expert users. Furthermore, *methods* describe sequences of operators in order to achieve sub-goals. Finally, *selection* rules are applied if more than one method can be used to achieve a goal. An overview of methods from the GOMS family with different application areas is given in [11].

The Keystroke-Level Model (KLM) is a simplified approach of GOMS for modeling human performance with an interactive application [5]. The focus of this analytical modeling method is to predict the interaction time of expert users for specific tasks. Larger tasks can be divided into smaller unit tasks. These unit tasks can further be subdivided into an acquisition phase and an execution phase. For the execution phase of each unit task, an expert writes down the method to perform the task and counts the keystrokes involved. By adding up the allocated times for these actions, the overall execution time can be predicted.

ACT-R [2] is a cognitive architecture which incorporates various theories about human information processing and has been applied to the analysis of HCI. While ACT-R implements the limits of human information processing (such as restricted working memory), it allows for parallel processing in some cases, making ACT-R models very powerful but complex compared to other approaches. Due to this complexity, usually it is only applied by experts in research.

Based on the aforementioned approaches, CogTool [12] was developed in order to simplify model-based evaluation and make it accessible to developers with less psychological expertise. It is based on a compilation approach which produces ACT-R models from KLM sequences. The KLM model is automatically generated from a user interaction demonstration which the designer performs on a storyboard of the UI. This storyboard consists of screenshots or mockups and needs to be created for each evaluation. User actions are then translated to perceptual and motor operators like *look-at* and *point*. *Think* operators that represent mental activity of the user are

automatically added at every decision point. The resulting model is compiled together with information from the storyboard into an ACT-R model which is used to predict task execution times. Initial work in applying CogTool to further interaction devices is described in [13]. While the results are promising, comparisons with real user tests also revealed that changes to the underlying KLM model need to be made in order to cope with specific domains [13].

To sum up, model-based evaluation can only make predictions on the basis of what is known about the user. Since some aspects of the users' interaction behavior, and in particular their judgments about the system (e.g. user satisfaction), cannot be anticipated by an analyst or a formal method, today's methods are not able to fully substitute evaluations with real users and experts. Instead, model-based methods should be used complementary to expert or user-based evaluations; e.g. to make early decisions between design alternatives or when user tests are too costly or too hard to apply.

PREDICTING TASK EXECUTION TIMES BY COMBINING UI DEVELOPMENT MODELS AND AUE TOOLS

When applying AUE to the development of model-based adaptive UIs, there are specific requirements that need to be addressed [23]. Basically, this involves making information about the application, the context of use, the user, and the task accessible to the specific AUE approach. In previous work, main benefits of combining MBUID with model-based evaluation were highlighted which lie in reducing the complexity and costs for applying AUE [22].

A specific approach, which deploys UI development models of the previously introduced MASP framework [3] to a model-based usability evaluation approach, has been introduced in [22]. The work in this paper builds on top of that basic approach and adds further integration of the UI development models and thus more automation to the evaluation process. However, we do not only address this extended automation but also demonstrate how specific requirements of an AUE tool can be covered by information from the UI development models in order to gain improved results.

Below, we give a short summary of the MASP framework models in order to better present required extensions to the architecture which are introduced in the following sections:

The Task Model defines the interaction flow between user and application. Interaction tasks define direct actions by the user and application tasks define application logic that is executed without user actions.

The User Interface Models are specified independently from modalities and devices on the level of the AUI model, while modality-specific definitions of the CUI models exist in parallel at runtime and are connected to the final UI on the specific devices.

The Domain Model serves as a dynamic storage for domain information. Relevant for rendering the UI is that this information is used to dynamically link domain data to UI elements for input and output.

The Context Model holds sensor and device data in order to reflect on the environment, user, and interaction devices.

Enhanced Model-based Architecture for Predicting Task Execution Times

As introduced in [22], for our combined approach we chose to apply CogTool as an AUE tool for predicting task execution times. The main reason for this decision is that CogTool is widely used and the underlying ACT-R concepts are sufficiently validated. In order to apply an evaluation with CogTool, representations of all visited UI elements from each UI screen that is on a demonstrated interaction path are required. For this reason, the task model of the application does not suffice alone. This is due to the fact that the evaluation with CogTool relies heavily on the interaction logic of the application in combination with UI surface information, such as type, label, size, and position of UI elements (see [23]). Consequently, the approach needs to provide means for deriving this UI information and – as equally important – a simulation component to extract required interaction paths. For this purpose a module was implemented that transforms UI information from the AUI and FUI models of the current UI screen and provides this to a user model for simulating interaction. The resulting interaction traces are then compiled to ACT-R using CogTool and can be analyzed further. Figure 1 gives an overview of the processes and participating models which are described in the remainder of this section.

Converting UI Information and Performing Interaction

In a first step, all UI elements from a given UI screen and their relevant attributes for an evaluation using CogTool are identified. The intermediate model *UI Element Information* from Figure 1 is used to provide this information from each UI element. During this process, we make use of the task models that are being executed in the MASP. We build a subset of the enabled tasks by filtering only tasks that are actually presented on the screen - the presentation task set (PTS). Each task of the PTS also links to an AUI element that has a FUI representation on the current UI screen. Consequently, all required FUI elements are then collected by following the mappings from AUI elements to FUI representations in the runtime platform (see upper left of Figure 1).

For the case studies of this paper, a concept was applied which uses expressions in the AUI models in order to map AUI elements with their corresponding HTML elements (on the FUI level). These mappings from AUI to FUI were implemented using descriptive paths to the location of the specific HTML element in predefined templates. By following these mappings, the required interconnection from task and AUI models to FUI elements is accessed during runtime. Hence, this concept provides up-to-date information of the UI screen as it is at any given point of time on any available interaction device and for any adaptation variant of the UI. The required attributes are then transformed to the aforementioned *UI Element Information* (arrows 1a and 1b in Figure 1).

During simulation the *User Model* is initialized with *User Task Knowledge* which consists of a list of labels that need to be interacted with in order to reach a specified goal state. This

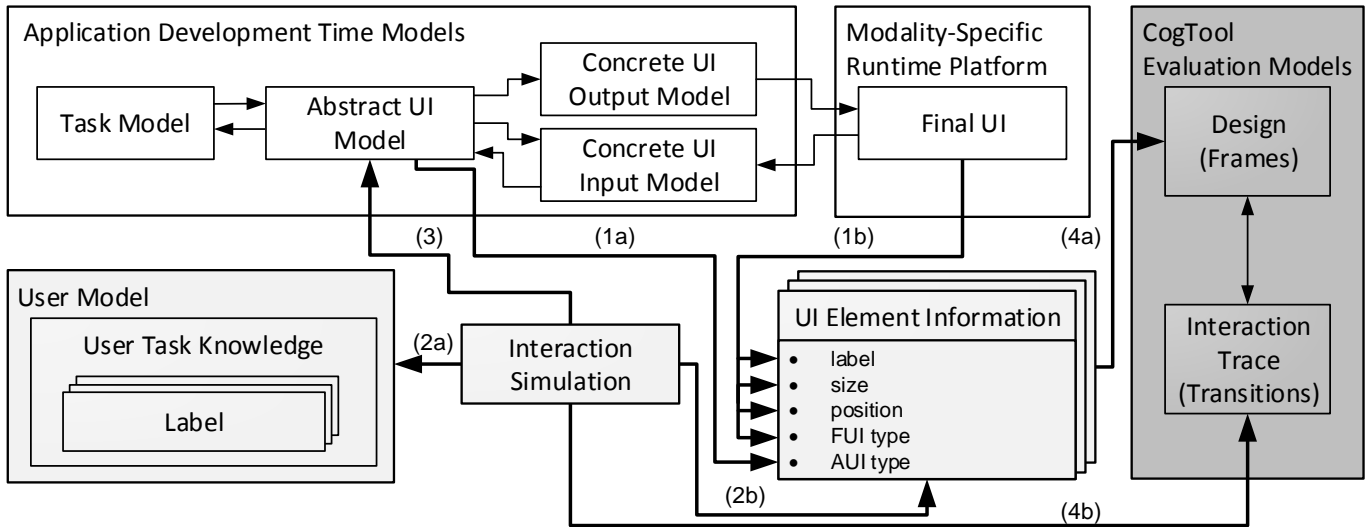


Figure 1. Integrated architecture with interconnections between UI development and runtime models (white) and intermediate models required for simulation (light gray) and evaluation (dark gray). Bold lines indicate processes and thin lines depict mappings and exchange between the models.

process is controlled by the *Interaction Simulation* component which accesses the *User Task Knowledge* and compares it to the labels from the *UI Element Information* of the current UI screen (arrows 2a and 2b in Figure 1). The definition of this knowledge matches the approach of CogTool to evaluate expert interaction tasks that are pre-defined by the developer and is used for the evaluation of the user studies in the following sections. A more detailed description how the labels are compared and alternative ways to retrieve required information for the *User Task Knowledge* are presented in [22].

When the current *User Task Knowledge* matches the label of a *UI Element Information*, an interaction on that specific element is simulated. This interaction is then performed on the corresponding AUI element which activates the next task set of the application (arrow 3 in Figure 1). By this means, the follow-up UI screen is triggered, which is evaluated accordingly until the goal-state is achieved; i.e. all labels from the *User Task Knowledge* are matched.

In the last step, each set of *UI Element Information* from the current UI screen and the simulated interaction are transformed to specific CogTool models - *Frames* that are interlinked by *Transitions* (arrows 4a and 4b in Figure 1). By this means an interaction trace is created and evaluated automatically in order to predict the task execution time. Additionally, the whole interaction process can also be validated using internal analysis methods and views of CogTool.

Below, we describe a pilot usability study that was conducted with a prototypical application of the MASP in order to gain first results from real user interactions that we used as a basis for enhancing and evaluating the described approach later on.

COGNITIVE USER MODEL

In order to test whether automatically generated cognitive user models can inform the usability engineering process, we conducted a usability study with a MASP based cooking assistant (CA, see Figure 2). The CA consists of several UI

screens and allows searching for recipes depending on attributes like calorie intake and type of dish. After having selected a recipe, the user can check the needed ingredients against the contents of the storage cabinet and – if applicable – generate a shopping list. The actual preparation of the selected recipe is performed using a step by step guide.

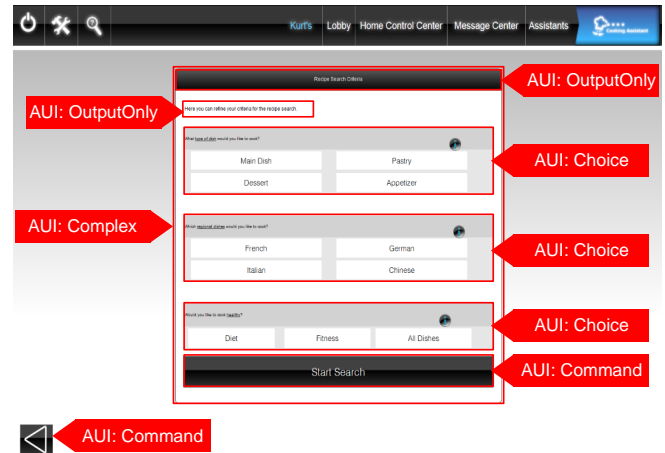


Figure 2. Screenshot of the English version of the cooking assistant with annotated AUI elements and their types.

Pilot Usability Study

The usability test presented here focused on the task of finding a recipe and took place in May 2013. Ten subjects (40% female; $M_{age}=29$, $SD_{age}=12$) were recruited mainly off campus. The CA was presented on a 19" (48.3 cm) touch screen with a 4:3 ratio mounted on a wall cupboard above the kitchen sink (see Figure 3). We recorded the interactions of the subjects with the user interface while they completed five simple tasks, e.g. "Search for German main dishes and select lamb chops".¹ All subjects had worked with the recipe finder and

¹The full instructions are available for download at <http://www.tu-berlin.de/?id=135088>

the touch screen in a previous part of the experiment so that they could be considered adequately trained.² Task instructions were given verbally, user actions were logged by the MASP and additionally recorded on video to be able to identify system errors or misinterpreted user actions (see placement of camera in Figure 3). The video recordings and system logs were synchronized using ELAN [26], which was also used to annotate user interaction errors, such as wrong and unrecognized clicks, as well as system response times, starts, and ends of individual trials.

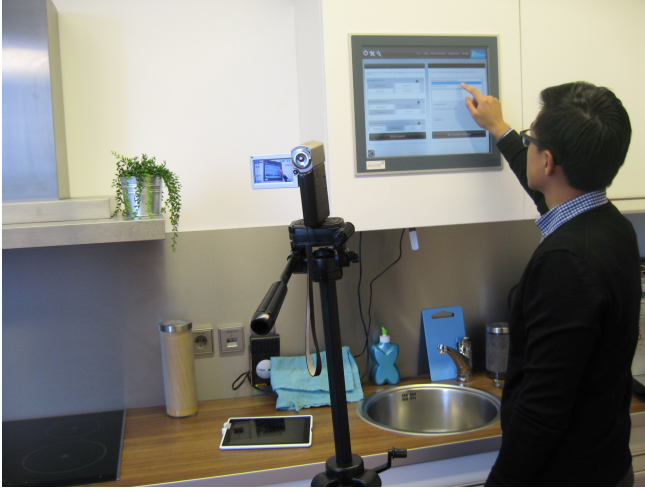


Figure 3. Experimental setup with the cooking assistant on the mounted touch screen for the pilot study conducted in May 2013.

Results

Average task execution times and number of errors per task are presented in Table 1. The relatively high number of errors is most probably caused by the six-year-old touch screen used during the experiment which suffered from a substandard input detection compared to current devices. As described above, these errors were extracted from video and log recordings. Errors are especially important to us, because CogTool, which is part of our combined approach, aims at predicting task completion times for expert users in error free conditions. For the comparison between our data and a CogTool model to be meaningful, we have to discard all erroneous trials beforehand. In case of the pilot study, this yields only one to four observations per task – too few for statistical analysis.

Task / Recipe	Steps	time (s)		errors	
		M	SD	M	SD
1 Lamb Chops	5	10.7	2.3	1.7	1.3
2 Roast Apples	5	8.7	2.4	1.0	.8
3 Panna Cotta	6	11.4	6.3	.4	.5
4 [Group size]	4	3.8	1.5	.8	1.1
5 Chicken Breasts	14	32.6	8.0	3.9	1.3

Table 1. Task completion times and error rates from the pilot study.

²The experiment was embedded in a larger evaluation study. We only report the relevant parts here.

In order to get as much information as possible from the experiment, we rejected complete tasks as subject of the analysis and went for times between pairs of clicks instead. If for example a task consisted of five clicks, and an error happened between click 2 and 3, we removed everything between 2 and 3, and used only the remaining three steps (1 to 2, 3 to 4, and 4 to 5).

Next we developed a classification for the different types of clicks. The class of a click should be related to the time a user needs to perform it. The simplest and fastest one should be repeated clicks on the same UI element. We will call this type *same button* in the following. The other extreme are clicks on buttons that are not part of the same UI screen, i.e. a new page must be loaded before the second click can be performed. We denote this as *new screen*. The remaining clicks are performed on the same form, but on different buttons. The buttons of the cooking assistant’s user interface are grouped semantically, e.g. there is a button group called “Regional Dishes” with individual buttons for “French”, “German”, “Italian”, and “Chinese” (see Figure 2). We decided to differentiate between clicks within and across those groups and finally obtained four types of click pairs, ordered by semantic and also physical proximity: *same button* (repeated clicks); *other button in the same group*; *other button in another group*; *other button on a later displayed new screen*.

In total we observed 447 single clicks during the experiment, of which 78 (17%) had to be discarded due to hardware (mainly touch screen) errors. The remaining clicks formed 218 valid pairs of clicks that could be divided into the four categories above. The time interval between clicks is significantly different depending on type (linear mixed model [20] with subject as random factor, $F_{3,205} = 19.9, p < .01$).

Visual examination of the data indicated the presence of extreme outliers (up to 10 seconds between clicks), most probably caused by the inclusion of erroneous trials in the analysis. We therefore based any further examination on robust statistics like 20% trimmed means [25].

Automatically Generated Cognitive User Model

Main goal of the initial study presented above was to show that automatically generated cognitive user models can provide useful information to system developers and designers. As a baseline, we exported the task information contained within the MASP UI models to CogTool as explained in the previous section (see also Figure 1). CogTool bases its predictions on the Keystroke-Level Model, estimating the motor time needed to perform a click by Fitts’ Law [6] and placing generic *think* operators (1.2 s) at every decision point (i.e. between clicks). This basic approach yields already moderate fits, but cannot really differentiate between all of the click types (see CogTool “Ootb” out of the box predictions, ■ in Figure 4). The goodness-of-fit statistics given in Table 2 indicate significant room for improvement. CogTool out of the box predictions are especially off the mark in the *same group* and *new screen* conditions. In the following part, we examine these conditions in more detail. We provide theoretical explanations for the mismatch between model and observed

data and derive a revised cognitive user model. At the same time, we demonstrate how the cognitive user model benefits from information contained within the development models.

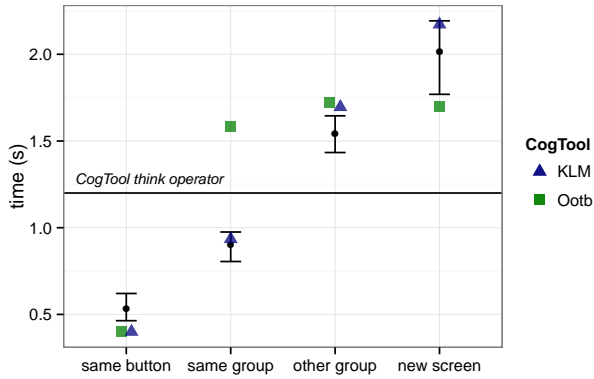


Figure 4. Average times between clicks on different elements on the user interface for the pilot study. Circles denote 20% trimmed means, lines bootstrap 95% confidence intervals (10000 repetitions).

Hypothesis I: Units of Mental Processing

The first observation worth discussing is the difference between the *same group* and *other group* clicks (see Figure 4). Buttons in the same semantic group are also physically closer to each other; therefore it should need less time to move the finger from one to another. But is this explanation sufficient? Fitts’ Law provides well-researched predictions for the time the user needs for finger pointing movements. In our case, Fitts’ index of difficulty is close to 1 bit within a group and raises up to 3 bit across groups. Even when using a comparatively high coefficient of 150 ms/bit, Fitts’ Law only predicts a difference of 300 ms between *same group* and *other group* clicks. This is much less than the 640 ms that we measured; hence we have to reject Fitts’ Law as a single explanation.

In addition, the absolute time between clicks within the same group in the pilot study was approximately 900 ms. If the premise of a generic 1.2 s *think* operator before every click holds, this would not be possible. We concluded that the basic CogTool model does not sufficiently match our data and decided to augment the model using additional rules and heuristics from the literature [10] and structural information of the user interface from the MASP user interface models [22].

The original formulation of the Keystroke-Level Model formed the basis for the revised cognitive model. Card, Moran, and Newell present several rules for the placement of *think* operators. In principle, those rules are already incorporated in CogTool, but some cannot be applied automatically, i.e. without human *interpretation* of the user interface in question. The rule that is most important for the given experiment is rule 2, which says that if several actions belong to a cognitive unit, only one *think* is to be placed in front of the first one. This rule definitely applies to the *same button* condition, where one button is clicked a fixed number of times in a row. A more interesting case is the *same group* condition which indicates consecutive clicks within a group of buttons belonging to the same concept, e.g. changing the type of dish

from “appetizer” to “dessert”. In terms of cognition, this task can be solved using a single chunk of memory that represents the target type of dish (“dessert”), and thus no *think* operator is to be assigned to the *same group* condition. This also fits well with the empirical mean being noticeably smaller than the 1.2 s *think* time.

A main benefit of the described approach is that such cognitive units can be automatically extracted from the UI development models. As described above, we make use of the mappings between the runtime FUI elements and the AUI model. An example is presented in Figure 2, which depicts a screenshot of the recipe finder dialogue with additionally annotated types of the corresponding AUI element. While the overall container has the abstract type *Complex*, it consists of several further AUI elements which directly relate to a specific task, namely *OutputOnly* for presenting text or graphics, *Command* to trigger application actions and *Choice* for presenting elements from which to choose. FUI elements of the same semantic group, such as the nationalities of the dishes, are modeled using the same AUI element, which is of the type *Choice*. Thus, they are also grouped by their semantics on the level of the AUI model. Consequently, the specific AUI element of each FUI element is queried during conversion to CogTool. In case several FUI elements share the same AUI element (besides *Complex* container elements), they are considered to be a cognitive unit and the generation of the CogTool model is altered by removing *think* steps prior to consecutive interactions on FUI elements from the same AUI element.

Hypothesis II: System Response Times

Another property of the Keystroke-Level Model that we can take advantage from is the inclusion of system response times. Navigation from one UI screen to another took approximately 500 ms on the hardware setup used in the pilot study, which is comparable to the difference between the *other group* and *new screen* click types. Following the original KLM rules, system response times that occur in parallel to *think* operators are only taken into account to the extent that they *exceed* the think time, i.e. a frozen system does block mouse and keyboard input, but does not block the mental preparation of the user [5, p. 263]. CogTool applies this rule out of the box, and as the 500 ms screen loading time is shorter than the 1.2 s *think* operator, CogTool does not predict the difference between the *other group* and the *new screen* conditions (see Figure 4).

We decided to deviate from the original rules here. When Card et al. formulated the Keystroke-Level Model, their users were solving tasks like typing commands into line-oriented text editors from the pre-graphical-user-interface era. Using the system response time for the memory retrieval of the next command to be executed makes perfectly sense in this scenario. Our graphical interface and selection of tasks are sufficiently different to call for a gentle refinement of the KLM. The biggest change is that the main bulk of user actions is no longer (blindly) typing on a physical keyboard, but finger-tapping on dynamic visual elements on the screen. While memory retrieval still plays an important role in this scenario,

searching the screen for the next button to press should be at least equally so. And as visual search is not possible when the graphical objects to be searched are not yet presented to the user, this kind of preparation for the next physical action is being delayed by a system response if this is accompanied by the screen being blank. We therefore decided to add the blank time to our cognitive model in the *new screen* condition.

Comparable measures have been taken by other researchers before, e.g. while applying the KLM to handheld devices [16]. There, system response times that are not shadowable by *think* operators were introduced by adding bogus widgets to the system mockups. As it is our goal to get rid of mock-ups and use real applications instead, we did not follow this direction.

Due to the fact that system response times may vary between different platforms and devices, these need to be measured or estimated once. However, here MBUID approaches that rely on executable models benefit from the fact that the evaluation of the models is actually directly combined with testing the real application. This specifically allows measuring and including system response times into the conversion process of the interaction traces. After each simulated user interaction it is checked whether the presentation task set has changed. If so, it can be assumed that a new UI screen was rendered. In this case the system response time is automatically added in sequence with a *think* operator because users cannot perceive information from the following UI screen until its rendering is finished. If the presentation task did not change, the system response time is added conforming to the CogTool implementation and might be shadowed by a parallel *think* operator. To sum up, task and AUI model are used to check on changes in the UI and to alter the cognitive user model.

Hypothesis III: Monitoring

What remains is the 360 ms difference between the *same button* and *same group* click types. Clicking the same button repeatedly does not incorporate movements of the forearm, moving the finger to another button of the same group does. Therefore we expect a difference between the two types. The movement time can be predicted using Fitts' Law, but again, this does not give sufficiently big estimates. We propose a monitoring hypothesis to fill this gap: Given the bad reliability of the touch screen, we assume that our subjects monitored whether their physical tap on the screen yielded a correct system response (i.e. a visible change of the displayed button). The time that this additional monitoring step needs consists of the time the systems needs to display a change and the time the user needs to notice this change. This system time is about 300 ms for our device. CogTool can be used to predict the time the user needs to encode and notice the change.

During conversion to CogTool the system response time (see Hypothesis II) and an additional *look-at* operator are added automatically in order to model a user monitoring if the desired action is reflected in the GUI. For this purpose, the FUI type of the UI element is queried and, in case it is an interaction on a UI element that can be toggled; e.g. radio buttons and checkboxes, the cognitive model is altered accordingly.

Model Fit

Predictions of CogTool out of the box (■, labeled "Ootb") and our augmented CogTool model (▲, labeled "KLM") are displayed in Figure 4. For both models, we computed several goodness-of-fit statistics, namely the coefficient of determination (R^2), the root mean squared error (RMSE), the maximum likely scaled difference (MLSD, [24]) and the maximum relative error (max diff). We based the comparisons on 20% trimmed means and bootstrapped confidence intervals. The results are reported in Table 2.

Model	R^2	RMSE	MLSD	max diff
CogTool Ootb	.597	0.39 s	4.6	75.9%
CogTool KLM	.995	0.13 s	1.4	25.3%

Table 2. Goodness-of-fit of the two cognitive models for the pilot study. Statistics are based on 20% trimmed means of the time intervals between the clicks of the users.

CogTool promises prediction "within 20% of the actual performance" out of the box [12]. We could not achieve this in the pilot study, the results are nevertheless promising. Especially the CogTool KLM model obtained a R^2 close to the maximum possible value of 1. The MLSD of 1.4 indicates that the measured differences between the model and the empirical trimmed means are very close to the amount of uncertainty in the data. The model therefore cannot be refined much more without taking the risk of overfitting.

Discussion

We showed that semi-automated predictions of task execution times based on UI development models are possible. On top of that, we identified properties of the MASP UI models behind our interface that we could use for further improvements of the cognitive user model. The goodness-of-fit that we achieved without parameter tuning is very promising. The use of few and very selective tasks and the small number of subjects put the validity of the cognitive user model into question, though. Thus, further analyses based on independent empirical test data are necessary.

Using CogTool for click by click analyses instead of predicting task execution times could be criticized as atypical application or even unfair towards CogTool. The 1.2 s *think* operator time used in CogTool is estimated from empirical data, and as Card et al. state, these operators for mental activity vary strongly within and across subjects. This means that the horizontal line labeled "CogTool think operator" in Figure 4 should not be considered a strong cut-off, but rather demarks a somewhat blurry transition area. When applying KLM to full tasks, the variance within think times should partially average out, i.e. a model with moderate fit on click level can still achieve good fit on task level. As the empirical basis of the pilot study is too weak for this kind of analysis, we decided to perform a second experiment.

EVALUATION

In order to test the validity of the cognitive model that we developed on the basis of the empirical data of the pilot study, we conducted a validation study with new subjects and additional user tasks. The study took place in November 2013,

12 subjects were recruited mainly from within the research groups of the authors (17% female; $M_{age}=28.8$, $SD_{age}=2.4$). In order to achieve a higher coverage of the cooking assistant’s functionality, each subject completed 34 individual tasks. The presentation of the user interface was moved from the wall-mounted touch screen to a personal computer with integrated 27” (68.6 cm) touch screen with a 16:10 ratio. This was done both to reduce error rates compared to the first experiment and to test whether the model generalizes well to new devices due to adaptation of the UI caused by a different aspect ratio and size of screen. The latter question is of high importance for automated usability evaluation of plastic UIs that are developed without knowing on which devices they will later be used [22].

Besides the changes in subject group, physical device, and task selection, we closely followed the experimental design of the previous study. User actions were again logged by the MASP and recorded on video. In the same way, system response times, start times and end times of the individual tasks were annotated using ELAN [26].

Results

We recorded a total of 180 minutes of video footage, about six times the amount of the pilot study. For being able to compare the results with the previous experiment, we conducted an analysis on the click-to-click level first. We observed a total of 1930 pairs of clicks that can be divided into the classification given above. Means and confidence intervals for these are given in Figure 5. The differences found in the pilot study are qualitatively replicated in the second study. We observed an overall increase in speed, though. The average time between clicks decreased from 1.41 s to 1.04 s between the experiments (linear mixed model [20], additional factors: click type as fixed and subject as random effect, $F_{1,20} = 55.5, p < .01$).

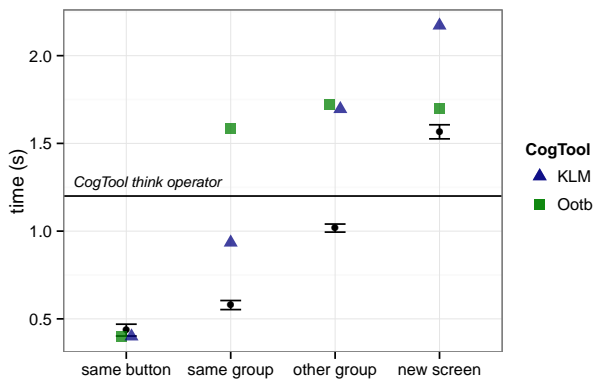


Figure 5. Average times between clicks on different elements on the user interface for the validation study. Circles denote 20% trimmed means, lines bootstrapped 95% confidence intervals (10000 repetitions).

For examining the generalizability of our cognitive model, we used the unchanged models from the previous study and computed predictions for the tasks of the new one. Goodness-of-fit statistics are given in Table 3. While the correlations es-

pecially between the KLM model and the data are still near to perfect, RMSE and relative difference degraded substantially.

CogTool Ootb	R^2	RMSE	MLSD	max diff
pairs of clicks	.425	0.61 s	20.0	173.9%
complete tasks	.735	3.32 s	8.4	104.0%
CogTool KLM	R^2	RMSE	MLSD	max diff
pairs of clicks	.927	0.47 s	13.8	66.0%
complete tasks	.965	2.77 s	8.4	70.9%

Table 3. Goodness-of-fit of the two cognitive user models for the validation study. Statistics are based on 20% trimmed means and bootstrapped confidence intervals (10000 repetitions).

Due to the higher number of subjects, tasks, and fewer errors, we could extend our analysis to the execution times of complete tasks instead of single clicks. Task completion times are a much more natural measurement than clicks and are also the level of analysis originally used by the Keystroke-Level Model and hence CogTool. We grouped the 34 tasks into seven categories by the total number of steps per task and by how many of these were *same group* and *new screen* type of clicks as those were the critical conditions in the first study. Average completion times and model predictions per category are given in Figure 6, goodness-of-fit statistics in Table 3.

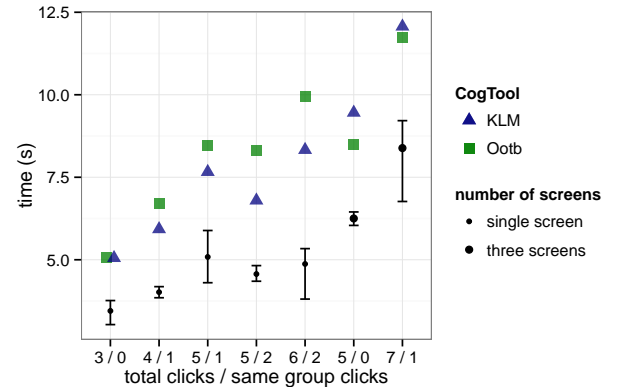


Figure 6. Average task completion times for the validation study. Circles denote 20% trimmed means, lines bootstrapped 95% confidence intervals (10000 repetitions).

Discussion

The results of the validation study are qualitatively similar to the first study, but we also found big quantitative differences. The overall gain in click speed between the experiments could be accounted to differences in subject selection, device used, length of the experiment and so on. As all of these variables are confounded; we can neither confirm nor reject any of them as influence factors.

What is more important is the degree of generalization of our cognitive model to the changed situation. The goodness-of-fit of the model to the new data is an important indication of the usefulness of our approach to automated usability evaluation in general. We will get back to this line of thought in the general discussion below and only discuss the evidence here.

Of the fit statistics that we applied, RMSE and maximum relative difference are most sensitive to overall shifts in the

data, whereas the determination coefficient R^2 neglects these and identifies changed relations between (classes of) observations. R^2 being very high in our case means that especially the KLM model describes the differences between the identified types of clicks very well. RMSE, scaled difference (MLSD) and relative difference on the other hand clearly show that our predictions miss the actual task completion times.

One promising result is that the modified KLM model still yields substantially higher fits than CogTool out of the box in the second study (see Table 3). This provides backing for the validity of the hypotheses that we derived from the results of the first study. The goodness-of-fit computed on task completion times being comparable to the one computed on click level also hints at the robustness of our approach.

Finally the most general question: Does the relatively bad fit put the KLM that formed the theoretical basis of our user model into question? Looking at Figure 5 shows that nearly all clicks moved below that 1.2 s *think* operator time in the validation study. This could be taken as evidence for the general inappropriateness of such an operator. While being correct in itself, this argumentation misses the heuristic nature of the KLM. As Card et al. pointed out, users differ a lot in how they mentally encode a task, and higher levels of user expertise can be modeled by placing fewer *think* operators for the same task [5, p. 265]. Taken together with the highly selective group of participants in the second study, this can explain the apparent disappearance of mental preparation times.

GENERAL DISCUSSION

With the help of information from UI development models and an AUE tool (CogTool) we have created a cognitive user model that yields very good fits to data from a usability study conducted in parallel. We validated the model with data from a second experiment where the task domain was kept constant, but user group and physical device were varied, thereby testing generalization to new contexts of use. While the cognitive model was able to predict time differences between the new tasks of the second experiment very well, we also observed an overall shift in task completion time that was not anticipated by the model. We can conclude that the model provides a good basis to compare different UI adaptation variants, or to predict the efficiency of a UI for frequently vs. rarely used functions. Especially in the domain of smart home environments, specific recurring functions are used quite often and profit from an efficient interaction. In addition, the validation study shows that adding information from UI development models significantly improves the goodness-of-fit of the cognitive user model.

Compared to further model-based evaluation approaches that focus on evaluations of task models (e.g. [19]), our integrated approach also uses AUI and FUI information and thus allows creating richer cognitive models. Especially, exact button positions (and their labels for matching purposes) can be used for optimized predictions using Fitts' Law [6] without requiring data from real user tests. It needs to be remarked that compared to approaches that focus on exhaustive task model evaluations, the degree of simulation is currently limited to predefined interactions. So, our approach presents a

complementary evaluation with a focus on detailed execution time predictions. But specifically this focus provides benefits when evaluating plastic UIs because task execution times are a quantifiable criterion of usability. When adapting the UI to a different context, the automatically predicted execution time, whether annotated during development or predicted during runtime, can e.g. serve as a hard constraint to exclude adaptations or to prefer a specific variant over others.

Specifically for the domain of model-based evaluation, our approach presents a benefit that is highly important. Modeling the system usually requires a significant part of the modeling effort [14]. Thus, connecting the user model to the real system and still using information from the development models simplifies model-based evaluation greatly. Finally, we showed that an additional benefit lies in a better placement of *think* operators which, otherwise, would require case-specific expert knowledge.

Possible areas of application arise, when taking a look at current challenges in MBUID. On the one hand, our approach provides a tool that can predict efficiency across different contexts of use and thus can be included in developing and maintaining plastic UIs. On the other hand, usability regression testing can be included into the development cycle after each iteration step, starting with early prototypes. The described approach could also be used to add automation to existing solutions [1]. Finally, the applied methods to gain information from UI templates in combination with dynamic CSS and JavaScript even allows automated extraction of UI information in the current trend of *responsive design*.

CONCLUSIONS

In this paper we have presented an approach for automatic generation of cognitive user models in order to predict task execution times during model-based UI development. The process strongly benefits from information available in the UI development models which we used to create better cognitive models without requiring further intervention by usability experts. The described approach does not require extensive knowledge in the usability domain in order to be introduced to evaluation cycles during the development process. Typical areas of application arise when there is a need to compare different design decisions or alternative adaptations such as when creating plastic user interfaces. Even if usability experts are available, they can base their work on the automatically created CogTool models, sparing them from the laborious creation of UI mockups and storyboards.

Acknowledgement: We gratefully acknowledge financial support from the German Research Foundation (DFG) for the project “Automatische Usability-Evaluierung modellbasierter Interaktionssysteme für Ambient Assisted Living” (AL-561/13-1).

REFERENCES

1. Abrahão, S., Iborra, E., and Vanderdonckt, J. Usability evaluation of user interfaces generated with a model-driven architecture tool. In *Maturing Usability*, Human-Computer Interaction Series, Springer London (2008), 3–32.

2. Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., and Qin, Y. An integrated theory of the mind. *Psychological review* 111, 4 (2004), 1036–1060.
3. Blumendorf, M., Lehmann, G., and Albayrak, S. Bridging models and systems at runtime to build adaptive user interfaces. In *Proceedings of the 2Nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '10, ACM (New York, NY, USA, 2010), 9–18.
4. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A unifying reference framework for multi-target user interfaces. *Interacting with Computers* 15, 3 (2003), 289–308.
5. Card, S. K., Moran, T. P., and Newell, A. *The Psychology of Human-Computer Interaction*. Erlbaum Associates, Hillsdale, New Jersey, 1983.
6. Fitts, P. M. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology* 47, 6 (1954), 381–391.
7. González-Calleros, J. M., Osterloh, J. P., Feil, R., and Lüdtke, A. Automated ui evaluation based on a cognitive architecture and usixml. *Science of Computer Programming Journal In Press* (05 2013).
8. Gram, C., and Cockton, G., Eds. *Design principles for interactive software*. Chapman & Hall, Ltd., London, UK, 1997.
9. Ivory, M. Y., and Hearst, M. A. The state of the art in automating usability evaluation of user interfaces. *ACM Comput. Surv.* 33, 4 (2001), 470–516.
10. John, B. E., and Jastrzembski, T. S. Exploration of costs and benefits of predictive human performance modeling for design. In *Proceedings of the 10th International Conference on Cognitive Modeling, Philadelphia, PA* (2010), 115–120.
11. John, B. E., and Kieras, D. E. The goms family of user interface analysis techniques: Comparison and contrast. *ACM Trans. Comput.-Hum. Interact.* 3, 4 (1996), 320–351.
12. John, B. E., and Salvucci, D. D. Multipurpose prototypes for assessing user interfaces in pervasive computing systems. *Pervasive Computing, IEEE* 4, 4 (2005), 27–34.
13. John, B. E., and Suzuki, S. Toward cognitive modeling for predicting usability. In *Human-Computer Interaction. New Trends*, J. Jacko, Ed., vol. 5610 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2009, 267–276.
14. Kieras, D. Model-based evaluation. In *The human-computer interaction handbook: fundamentals, evolving technologies and emerging applications*, A. Sears and J. A. Jacko, Eds. Lawrence Erlbaum Associates, Mahwah, NJ, 2007.
15. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and López-Jaquero, V. Usixml: A language supporting multi-path development of user interfaces. In *Engineering Human Computer Interaction and Interactive Systems*, R. Bastide, P. Palanque, and J. Roth, Eds., vol. 3425 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, 2005, 200–220.
16. Luo, L., and John, B. E. Predicting task execution time on handheld devices using the keystroke-level model. In *Extended Abstracts Proceedings of the 2005 Conference on Human Factors in Computing Systems*, G. C. van der Veer and C. Gale, Eds., ACM (Portland, OR, 2005), 1605–1608.
17. Mori, G., Paternò, F., and Santoro, C. Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Trans. Softw. Eng.* 30, 8 (2004), 507–520.
18. Nielsen, J. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
19. Paternò, F. Model-based tools for pervasive usability. *Interacting with Computers* 17, 3 (2005), 291–315.
20. Pinheiro, J., Bates, D., DebRoy, S., Sarkar, D., and R Core Team. *nlme: Linear and Nonlinear Mixed Effects Models*, 2013. R package version 3.1-113.
21. Polson, P. G., Lewis, C., Rieman, J., and Wharton, C. Cognitive walkthroughs: a method for theory-based evaluation of user interfaces. *International Journal of Man-Machine Studies* 36, 5 (1992), 741 – 773.
22. Quade, M., Lehmann, G., Engelbrecht, K.-P., Roscher, D., and Albayrak, S. Automated usability evaluation of model-based adaptive user interfaces for users with special and specific needs by simulating user interaction. In *User Modeling and Adaptation for Daily Routines*, E. Martín, P. A. Haya, and R. M. Carro, Eds., HumanComputer Interaction Series. Springer, London, 2013, 219–247.
23. Quade, M., Rieger, A., and Albayrak, S. Requirements for applying simulation-based automated usability evaluation to model-based adaptive user interfaces for smart environments. In *Distributed, Ambient, and Pervasive Interactions*, N. Streitz and C. Stephanidis, Eds., vol. 8028 of *Lecture Notes in Computer Science*. Springer, Berlin, 2013, 235–244.
24. Stewart, T. C., and West, R. L. Testing for equivalence: a methodology for computational cognitive modelling. *Journal of Artificial General Intelligence* 2, 2 (2010), 69–87.
25. Wilcox, R. R. Comparing medians: An overview plus new results on dealing with heavy-tailed distributions. *The Journal of experimental education* 73, 3 (2005), 249–263.
26. Wittenburg, P., Brugman, H., Russel, A., Klassmann, A., and Sloetjes, H. ELAN: a professional framework for multimodality research. In *Proceedings of LREC*, vol. 2006 (2006).