Adapting Writing Support Functions for Different Language Learners

Bowen Hui
Department of Computer Science
University of Toronto
bowen@cs.utoronto.ca

1 Motivation

Tutoring software is great for providing individualized attention that language learners need. Not only can software provide abundant exercises and examples for the learner, but instructions, explanations, as well as the learning environment can all be tailored according to the level of the student. A central problem that intelligent tutoring systems (ITS) address is how instructions and explanations can be constructed in an understandable way for the student. In some cases, the system's behaviour is based on a gross categorization of the student – whether the student belongs to the beginner, intermediate, or advanced category [8]. Unfortunately, these categories do not always capture how individuals from the same category have varying needs. Thus, students who are, for example, good at word choice but poor at vocabulary variation would be treated the same as students who are good at vocabulary variation but poor at idiom usage. The strength of ITS is the ability to provide support for individualized learning. Therefore, the categories used in determining how the system tailors its behaviour toward the student should be as fine-grained as possible. We believe that modeling these categories with low level skills is a step toward this direction.

Some recent work emerged in trying to combine user modeling with utility theory to account for the uncertainty of the user [1, 3]. However, the underlying model of these systems is a set of plans that describe possible user goals and actions. While modeling uncertainty is crucial, the use of plans restricts the flexibility and the scalability of the models. What would be great is if the adaptability considers the user's knowledge as well as the behaviour toward the system, but without being fixated on deterministic plans. This motivates our design methodology – using user knowledge as well as his/her on-going interaction with the system to determine the appropriate adapted version for the user. We first consider different aspects of a user model to construct a user profile. Based on these aspects, we design and select a customized version of a software that is most suitable for a particular user profile. Moreover, as the learner's skills improves and preferences change, the software will adapt its functionality according to the new profile.

In this paper, we consider the design of functionality support for language learners who want to write documents. In particular, we focus on the task of writing English documents and users who are learners of English as a second language (ESL). We begin by describing our general model of adaptability in Section 2. In Section 3.1, we discuss what relevant information from the user is needed in order to properly design a suitable writing tool, and we identify different customizable components that are suited for users with varying skills and preferences in Section 3.2. Rather than overwhelming the user by giving them everything in a "bloated" system [7], we attempt to derive the most appropriate subset of functions for a given user profile. Furthermore, we make the observation that the criteria used to generate design alternatives are intimately coupled with those for evaluating the software. Thus, we show how to derive metrics for evaluating the utility and usability of our software based on the elicited information in Section 3.3. Various design alternatives are described in Section 3.2, along with our evaluation plan in Section 4. Finally, in Section 5, we summarize our work and discuss ideas for future directions.

2 Using Adaptivity

Adaptive systems have received a lot of attention in artificial intelligence (AI) and human-computer interaction (HCI) communities [4, 7, 2]. A goal of adaptive systems is to automatically tailor various components of the system's functionality or presentation structure toward a user's preferences. An example in use is ITS where the system acts as the tutor and revises an explanation of a concept based on its assessment of the user's current level of understanding. Some adaptive systems also recognize certain pre-defined usage patterns so that the user can avoid carrying out the intermediate steps by allowing the system to do it on his/her behalf. A common example is a phrase completion box that pops up above the cursor in a document writing application. In this example, the completion phrase would pop up if the user types in a sequence of letters that has high probability association with a specific phrase, such as the name or the affiliation of the user who owns the software license. Both kinds of adaptive behaviour intend to unfold the complexity in today's ubiquitous software so that more people can use them easily and effectively. It is especially important for learners, whether it be language or computer novices, that using a system helps them by making their current task easier and not inducing more cognitive load in the process. However, different aspects of a task contribute to the easiness at varying degrees for different people. Therefore, personalization of software is crucial to designing effective software.

While AI research pushes for automated *adaptive* systems and HCI argues for user-controlled *adaptable* systems [6], we believe that the two areas have an essential process in common – the identification of customizable components in a software. Many systems offer methods to make software personal to users, but little research has focused on developing systematic approaches to identifying or designing these customizable components [5]. Our point of view in designing adaptive systems is first to analyze *what* functionality to make customizable and then use techniques from AI, HCI, or both, to tailor these components to the user at run-time. Upon systematically eliciting relevant information from the user, we generate design alternatives which can then be selected by the user or adapted toward the user's changing skills over time. We discuss these steps in detail in Section 2.1. In this manner, the software evolves along with the user's needs at a particular time. The adaptation of the software at run-time is discussed in Section 2.2.

2.1 Customization Method

It has been argued that design attributes should be based on user goals, user skills, and user preferences collectively [2]. The methodology we propose adopts this view, as we discuss how the three kinds of user requirements contribute to the identification of adaptable components in a software. These user requirements together form the variables of a *user profile*. Note that such a profile may capture the requirements for an individual user or a group of users with common interests. The key point is that the profile identifies the adaptive parameters. Thus, we begin by identifying a target user population, and carry out the following steps:

1. Elicit User Information

(a) given a target user group, identify their goals

(b) decompose the goals into manageable subgoals to obtain a *goal analysis graph*

(c) identify technological support (either full applications or functionality of an application) that can satisfy certain goal nodes

(d) elicit relevant skills for the goal nodes at the leaf level of the graph

(e) elicit preferences with respect to the overall goals in the graph (both environmental and technical factors)

2. Derive Design Alternatives

   (a) identify ways for technological support for each goal in the graph

3. Select Suitable Version

   (a) create user profile (individual or groups of users)

   (b) describe the impact of each function toward the skill and preference variables

   (c) choose the most feasible combination of components as the design alternative

The inner-workings of these steps will be illustrated in Section 3 in the context of an editor application. Full details of this methodology is elaborated in [5].

2.2 On-Going Adaptation

Once a selected version of the software has been distributed to the user, the software should continue to update itself according to the changing skills and preferences of the user. The need of adaptation over time is necessary to maintain user satisfaction of the software. Based on the elicited information in the user profile gathered at design time, we build a utility function so that the system evaluates its own usefulness with respect to what it observes about the user. The utility is based on the user's relevant skill variables, observations about the usage of various functionality of the system, as well as the user's preference toward the system's adaptivity. Thus, each functional component (e.g., spell check) removes itself from the interface if its utility is low and makes itself available to the user if its expected utility is high and the user's current skills are in need of help. In this manner, a version of the software is viewed as a subset of the available functionality of the underlying application. An example of how a component evaluates its utility is demonstrated in Section 3.3.

3 Text Editor Example

In this section, we use the example of an editor program to illustrate the role of adaptivity in designing software at design time and run time. The target user group in this context is adult ESL learners.

3.1 Elicited Requirements

We restrict our attention to the goal of composing English documents in this example. We begin by considering the relevant skills and elicited preferences from Step 1 of our methodology. The skills presented here are derived from psychology literature [9, 10] and the preferences are gathered empirically from non-native English speakers.

- goal: compose English documents

- skills:
  - using the software: visual perception, attention, memory, fine-motor control, typing speed
  - writing the document: spelling, word choice, wording (idioms, phrases), vocabulary variation, grammar, syntactic variation, proper referencing, structural organization
- preferences: improve writing speed, increase vocabulary variation, easy to ask for help, independent, better at making arguments

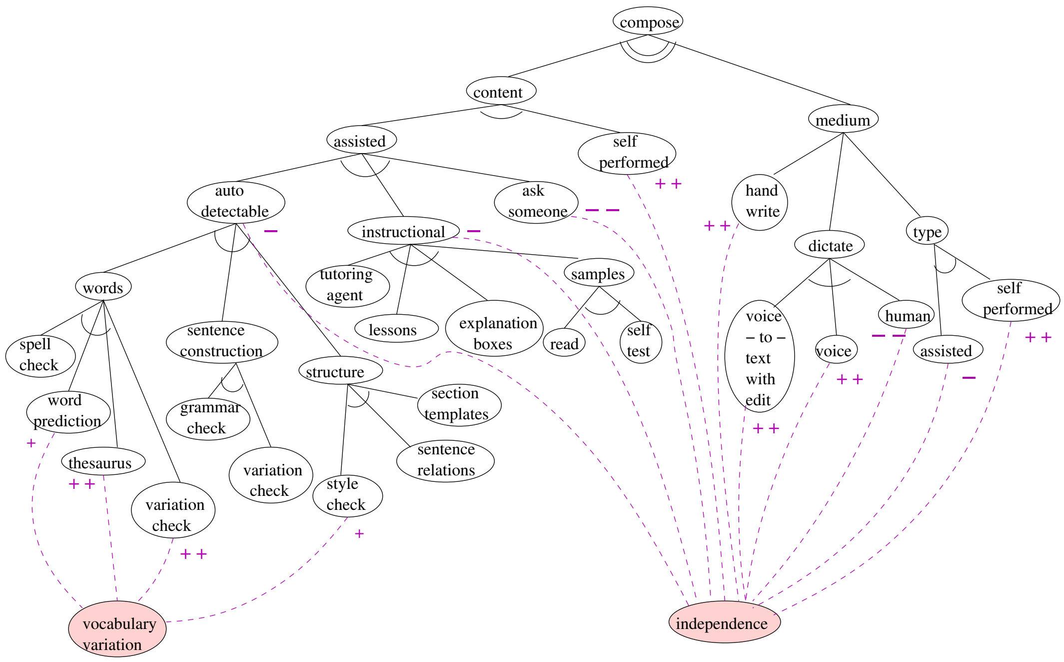

Figure 1: Goal Model for Composition

The graphical model that depicts the above information is shown in Figure 1 for the goal of composition. This model decomposes the goal `compose` into two necessary subgoals, `content` and `medium`. That is, a way to fill in the content of the composition *and* a medium to create the composition must both be satisfied in order for `compose` to be satisfied. From there, `content` can be achieved in a variety of ways via computer assisted functions (`assisted`) or purely through the user's own abilities (`self-performed`). The rest of this graph can be interpreted similarly. A double arc indicates that all subgoals must be satisfied in order for the parent goal to be satisfied, while a single arc denotes that satisfying any one subgoal satisfies the parent goal.

Another interesting aspect of this model is the role of preferences. Figure 1 shows two preferences regarding composition – `vocabulary variation` and `independence`. Although achieving good writing style is a bonus, people often display a strong preference toward using their own skills to achieve their goals. Thus, various modes and medium of composition have positive or negative contributions toward the `independence` preference. The degree of contributions are shown as double or single plus and minus signs respectively.

3.2 Deriving and Matching Design Alternatives

Notice that the various assisted methods under `assisted` in Figure 1 are in fact ways that technology can help the user. Thus, decomposing the high-level goals carries out the step of deriving design alternatives in Step 2 of our methodology. The decomposition shown in Figure 1 is not exhaustive due to continual technological advances. Nonetheless, the model is an expressive way of displaying alternatives as well as positive and negative impact of each solution.

What remains is the formulation of a user profile and matching it with the various functionality shown in the graph. Let us elaborate on the language skill variables in a user profile. We propose that these variables be discrete and range from -2 meaning very poor, to 2 meaning very good, to represent the competency level of the skill[1]. We extract the following language-related skill variables from the elicitation results presented in Section 3.1: spelling, word choice, phrase usage, vocabulary variation, grammar, syntactic variation, references, and structural organization. This gives us an integer vector of size 8 to describe the language background of each user. To annotate the contribution of each function toward a particular skill variable, we place each skill variable into the graphical model and connect them with a function that supports it. Moreover, a link with double plus signs denotes direct positive contribution, a link with a single plus sign denotes indirect positive contribution, while no links denote neutral contribution. (It is conceivable to have negative contributions as well, but there are none in this example.) These contributions are illustrated in Figure 2.

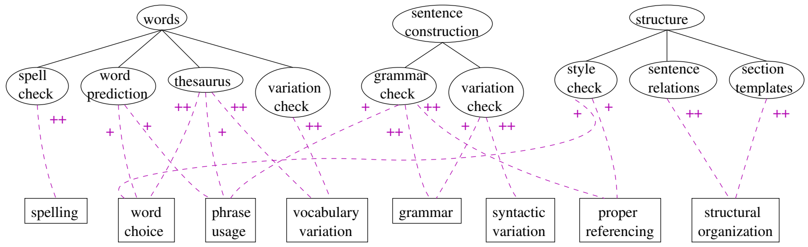

Figure 2: Skill Contributions to Auto-Detectable Functions

Other parts of the goal decomposition in Figure 1, such as the functions under `instructional` and the options under `medium`, can be decomposed similarly. Now, matching a user profile to a particular design alternative is simply an optimization of the supported skills and preferences of the user with the functionality that provides maximal contribution in those areas.

3.3 Utility Metric

Once a suitable version has been selected and distributed to the user, the system should maintain the user profile and adapt its functionality accordingly. We view the changing of system functionality as an adaptation problem – each system component uses a utility function to evaluate its usefulness for the user. Thus, the adaptation role for a component is to make itself available to the user or remove itself from the current version.

We use an evaluation function to adjust the adaptability of each component. For example, consider the word prediction component. Rather than always making predictions as the user types, predictions are only offered to the user based on the user's skill level and attitude toward adaptability. Intuitively, the level of adaptability behaves as follows:

[1]Educational and psychological tests can be used to assess the skill level of an individual.

- decrease if user never considers its suggestions
- decrease if user's corresponding skill improved
- stay off if the user turns it off
- offer suggestions if the user is really stuck
- stay the same if user considers and sometimes accepts suggestions
- increase if user always accepts its suggestions
- increase if user's corresponding skill worsens

We illustrate how this is formalized for the word prediction component. Let *level* be the level of adaptability which ranges between $[0, 1]$. That is, when *level* is closer to 1, the system offers more suggestions. Then, *level* is computed as follows:

$$level = tol * \beta * (usage - skill)$$

which consists of three localized functions. We define

$$tol = 1 - \frac{OFF}{ON + OFF}$$

where OFF is the number of times the function was turned off by the user and ON is the number of times the function was turned on. The function

$$usage = u_1 CONS + u_2 ACC + u_3 ON$$

where $u_i \in [0, 1]$ are weights, $CONS$ is the number of times a suggestion was considered, and ACC is the number of times a suggestion was accepted. Finally,

$$skill = s_1 SPELL + s_2 WCHOICE + s_3 VAR$$

where $s_i \in [0, 1]$ are weights, $SPELL$ is the level of the user's spelling skill, $WCHOICE$ is the level of the user's skill at choosing words, and VAR is the level of the user's word variation skill. All the usage variables are non-negative integers and the skill variables range from $[-2, 2]$.

The level of adaptability is defined such that, if user is not tolerant, then *level* goes down immediately. Otherwise, we consider the user's usage pattern as well as skill level. There, if the relevant skills are sufficient then *level* decreases slowly, at rate β, even if usage pattern shows that user likes the suggestions. The reason we still lower *level* in this case is because we want to support the independence of the user and ensure that the user does not overly rely on the system. On the other hand, if the relevant skills are poor, and usage is low, then we increase *level* slowly. Finally, if the relevant skills are poor but usage is high, that means the system is not actually helping the user. In this case, we decrease *level* slightly to support independence and encourage active learning on the user's part.

Other components of the system can be formulated similarly so that each component has its own utility function. The subfunctions *tol*, *usage*, and *skill* are reused in our user study to assess the usability of the system. In this manner, we use the same function for the system to evaluate its own utility and for people to evaluate the system's usability.

4 User Study Design

In a learning environment, the speed of performance should not be the ruling criteria in judging whether a technique is superior over another. What is also important is the learning process – whether the learned material can transfer to another domain and whether the learner is satisfied with his/her own progress. Thus, to truly assess these values in a writing support application, a longitudinal study is necessary to evaluate the effectiveness of the software. For now, we describe our plans for short-term usability studies.

We plan to collect user feedback for the customized alternatives as well as the adaptability of the system. To begin, we will consider three distinctively different user profiles. We plan to have five non-native English participants for each profile. Each participant will be asked to perform the same task with three software versions: a regular editor program of their choice, our editor with customized functions, and our editor with customized *and* adaptive functions. The participant will be given sometime to familiarize himself/herself with the interface and then s/he will be asked to perform two tasks. The first task is to type in a short letter verbatim while the second task is to freely compose a short letter to a friend on a specific topic. At the end, each participant will be asked to fill in a usability questionnaire. The questionnaire will specifically ask about the user's attitude toward the software and whether the user believed the software helped or improved the writing experience.

If the results of the user study confirms that customized functionality is better than today's standard editors, then our view of adaptivity in tutoring systems and the methodology for designing software would prove promising. On the other hand, if the results are negative, then we can use them to identify problems with the methodology – perhaps we are missing an important skill component, or ordering of preferences and trade-offs are not calculated optimally. Furthermore, the usability results can also be used to adjust the parameters of the utility function so that the adaptivity of the functions are more appropriately suited for those user profiles.

5 Concluding Remarks and Future Directions

A framework for adapting software for the user was presented in this paper. We discussed how to generate customized functionality for users with varying skills and preferences and illustrated how each component evaluates its utility for an evolving user profile. Our framework was demonstrated by designing an editor program for adult ESL learners. A user evaluation to provide feedback on our model is pending.

Currently, we are investigating in a Bayesian approach to model the uncertainty in the user's behaviour and the user's skills level. These variables should be modeled in a framework without dealing with fixed plans of goals or actions. We believe that a Bayesian approach would model the user more accurately, and thus, adapting the software more appropriately. In addition to tutoring systems, we hope to explore our framework with interactive machine translation (MT) systems. Making MT more adaptive would ideally reduce the human translator's cognitive load by suggesting useful completions and alternative words and phrases in the target text.

References

[1] S.M. Brown, E. Santos Jr., and S.B. Banks. Utility theory-based user models for intelligent interface agents. In *Proceedings of the 12th Canadian Conference on Artificial Intelligence (AI'98)*, 1998.

[2] P. Brusilovsky. Adaptive hypermedia. *User Modeling and User Adapted Interaction*, pages 87–110, 2001.

[3] C. Conati, A. Gertner, and K. VanLehn. Using bayesian networks to manage uncertainty in student modeling. *International Journal of User Modeling and User-Adaptive Interaction*, 2000.

[4] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The lumière project: Bayesian user modeling for inferring the goals and needs of software users. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, July 1998.

[5] B. Hui, S. Liaskos, and J. Mylopoulos. Requirements analysis for customizable software: A goals-skills-preferences framework. In *Submitted to Requirements Engineering*, October 2003.

[6] P. Maes and B. Schneiderman. Direct manipulation vs. interface agents: a debate. *Interactions*, IV:6, 1997.

[7] J. McGrenere. *The design and evaluation of multiple interfaces: A solution for complex software*. PhD thesis, Department of Computer Science, University of Toronto, Toronto, Canada, 2002.

[8] L.N. Michaud, K.F. McCoy, and C.A. Pennington. An intelligent tutoring system for deaf learners of written english. In *Proceedings of the 4^{th} International ACM SIGGRAPH Conference on Assistive Technologies (ASSETS 2000)*, November 2000.

[9] S.K. Reed. *Cognition: Theory and applications ($5^t h$ ed)*. Wadsworth, 2000.

[10] D.R. Shaffer, E. Wood, and T. Wiloughby. *Developmental Psychology: Childhood and Adolescence*. Wadsworth, 2002.