Requirements Analysis for Customizable Software: A Goals-Skills-Preferences Framework

Bowen Hui Sotirios Liaskos John Mylopoulos
Department of Computer Science
University of Toronto
{bowen,liaskos,jm}@cs.utoronto.ca

Abstract

Software customization has been argued to benefit both the productivity of software engineers and end users. However, most customization methods rely on specialists to manually tweak individual applications for a specific user group. Existing software development methods also fail to acknowledge the importance of different kinds of user skills and preferences and how these might be incorporated into a customizable software design. This paper proposes a framework for performing requirements analysis on user goals, skills, and preferences in order to generate a customizable software design. We illustrate our methodology with an email system and review an on-going case study involving users with traumatic brain injury.

1. Introduction

Software development has generally adopted a "one-size-fits-many" model in which software is designed for a few classes of user groups, rather than tailoring it to the needs of particular users [4]. Clearly, different individuals with different levels of expertise, different preferences, needs and goals require different kinds of services. For example, automated telephone banking might be designed in such a way that it is usable and effective for both business analysts with sophisticated profiles as well as students with basic needs. Also, a train scheduling system might be designed to have easy-to-understand instructions and interpret questions from foreign and local tourists alike. Likewise, survivors from brain injuries or other cognitive impairments might be able to continue their regular daily tasks, such as email and document composition, with the help of customizable software that is specifically tailored to support their individual skills and needs. Such ubiquitous situations create a demand for *software customization* – also known as software variability – a research problem that has received little attention in the literature.

Several sub-fields in Computer Science have begun to tackle this problem – each from its own perspective – including software engineering (SE), human-computer interaction (HCI), and artificial intelligence (AI). In SE, one approach to customizable software is to pack all possible functionality into a single generic system (e.g., MS Word). Unfortunately, excessive functionality makes for "bloated" software and perplexed users who cannot understand how to relate their needs, skills, and preferences to the feature space of the software [11]. Similarly, program families [10] attempt to define the set of allowable alternatives for a given set of features included in a family. Again, this approach offers a design-level perspective on the set of alternative customizations supported by a program family, and does not easily translate to user needs, skills, and preferences. Work in HCI has focused on how users like to be in control of the software and has developed intermediary mechanisms that allow users to directly manipulate the functionality and interface of a software system [11, 2]. At the other end of the spectrum, research in AI has developed automated techniques for monitoring user actions in order to infer user goals and preferences, which result in a revised user interface [8, 7].

In this paper, we propose a methodological framework for identifying customization parameters during requirements analysis. We argue that any customization model should encompass user goals, skills, as well as preferences that together form a foundation for customizable software from a user perspective.

The rest of the paper is structured as follows. Section 2 introduces our framework of goals, skills, and preferences and shows how to map these into design alternatives. To illustrate the breadth of alternatives that can be accommodated by our framework, and also to explain its inner workings, we apply it to a generic email program. Although this application is "simple" to use (at least to a researcher), it can often overwhelm an untrained adult, especially one with a cognitive impairment. Once the framework generates a set of alternatives, these can be mapped to a software architecture, as discussed in Section 3. That section also describes how customization can be carried out at run-time through the selection of a new set of modules that constitute the running system. In Section 4, we describe our experiences in applying the framework to an on-going case study involving users with traumatic brain injury (TBI). Finally, we summarize our work and highlight directions for future research.

2. Customization Methodology

Our proposed framework involves three kinds of requirements: user goals, skills, and preferences. We believe that the design of software should adopt a user goal perspective, asking questions such as "What tasks need to be performed if the user wants to fulfill certain goals?", "What if the user lacks certain necessary skills to perform these tasks?", and "How can the software be tailored towards what the user prefers?" These are the underlying questions that motivate the three components of our framework.

The foundation of our customization model is a detailed goal analysis on potential users for a particular application. Goal analysis has been found to be a useful modeling technique for analyzing and decomposing a problem in order to generate software requirements [17, 13, 3]. In designing software, we can adopt goal analysis to discover design alternatives by modeling target users as stakeholders and decomposing their goals to the level where they can be fulfilled by a *task* to be performed by a stakeholder or the system-to-be. Our proposal also accommodates user needs (e.g., communication) as well as desires (e.g., pass time) that improve one's quality of life, since both of these can also be cast as user goals. Furthermore, different people have different *skill sets* (e.g., computer fluency, eagerness to learn, memory deficits, language literacy) that define the amenities and obstacles that users have in using a software system. In particular, we adopt a set of cognitive skills described by Reed [14] into our model. In addition to goals and skills, users have *preferences* over how desired goals are to be fulfilled. Such preferences need to be identified by requirements engineers, modeled in the software design, and made available to the user at run-time for possible adjustment.

Our ultimate goal is to discover and document systematically the alternatives that can accommodate a

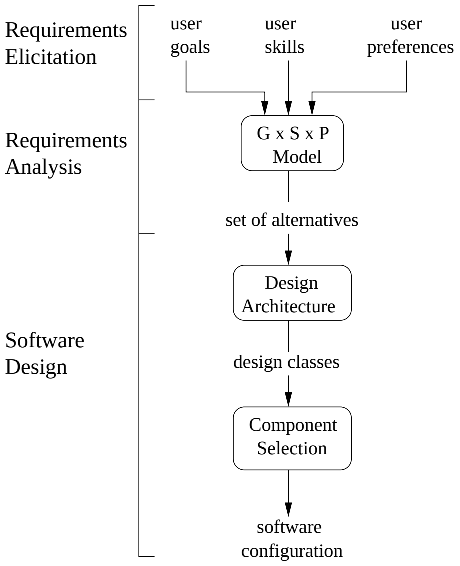

Figure 1. Methodology In SE Life Cycles

set of users goals. We believe that goals, skills, and preferences form the fundamental dimensions in formulating the requirements analysis problem from a user perspective. These dimensions are discussed in detail with our email example in Sections 2.1 to 2.3. In the analysis phase, our framework takes requirements as input and generates a set of ranked alternatives for the design phase. An alternative is defined as a set of tasks that together fulfill a set of target goals. Possible alternatives can be ranked against a user profile. Each alternative corresponds to a set of software components forming a particular architecture. This architecture and the selection of components at run-time are described in Section 3. Our overall framework is depicted in Figure 1.

2.1. Stakeholder Goals

The first step in modeling user goals is to identify them through elicitation techniques, such as running focus groups or questionnaires. We then analyze the elicitation outcome to generate a set of initial goals, which are then refined using goal analysis [17, 13]. Each goal is cast as a node in a goal graph, and is decomposed into subgoals according to the ways that the goal can be fulfilled. Subgoals of the same parent may participate in an AND relationship indicating that they all must be satisfied in order for the parent goal to be satisfied. Subgoals may also participate in an OR relationship indicating that if one or more of them are satisfied then the parent node is satisfied. The decomposition continues until we arrive at subgoals that can be satisfied by a person or by a system function. Such leaf nodes of the graph is referred to as *tasks* that can be carried out by the system-to-be or a stakeholder.

A subtree headed by the root node and containing nodes that together satisfy the root node is defined as an *alternative*. Identifying alternatives in this regard allows us to use goal analysis to diagrammatically develop design alternatives for software that accommodates a set of elicited requirements.

Consider the following example. During elicitation, we find that our potential users want to maintain contact with their friends and to keep in touch with people they know. However, through consideration of how these two goals may be achieved, we decide that they refer to the same underlying concept. Thus, we rename the goal as `Maintain Contact` and decompose it to identify proper software functionality to support it. (Please refer to Section 4 to see a partial decomposition of this goal.) During one of these iterations, we decide that sending email constitutes one way people can maintain contact. Furthermore, in order to compose an email message, one must choose a way of composing its contents and an input medium for the software. Figure 2 shows this binary decomposition, with the subgoals continuing in Figures 3 and 4 respectively. Figure 3 identifies six leaf nodes for expressing message contents, where the OR arcs denote that any subset of these nodes may be combined to define an alternative. Similarly, Figure 4 depicts different software input media.

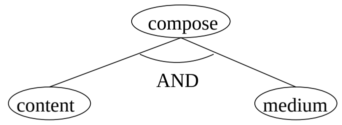

Figure 2. Goal Analysis for `Compose`

Several examples of alternatives are shown in Figures 5 to 6. Figure 5, on the left, shows the generic method of a user coming up with the content and entering the information herself. Other alternatives include choosing a combination of assistive functions for composing email content, as in alternative B on the right in Figure 5, and choosing a combination of assistive functions for email input, as in alternative C on the left in Figure 6. A very different alternative to A involves always asking for help from someone else. This is alternative D, illustrated on the right in Figure 6.

Due to the semantics of AND and OR relationships, goal analysis can result in an exponentially large space of alternatives. In particular, the number of alternatives defined by a goal graph depends exponentially on the number of OR relationships on that graph. The rest of this paper leaves complexity issue aside and concentrates instead on design ones. Section 4 reports on our experiences with goal graphs which contain millions of alternatives.

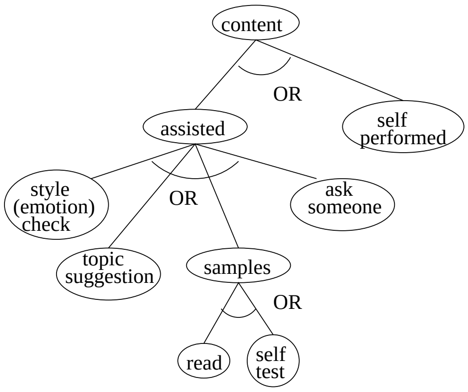

Figure 3. Goal Analysis for `Content`

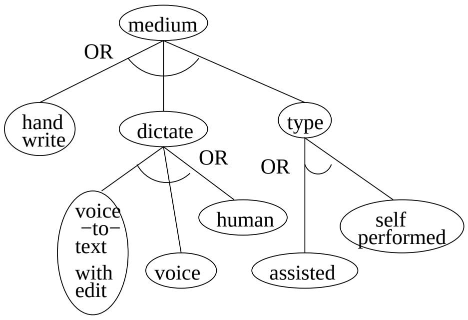

Figure 4. Goal Analysis for `Medium`

2.2. Required Skills

The second component of the model identifies the set of required skills for each function and uses them to rank various design alternatives. In developing a model for software skills, we collected background material from various areas of psychology including social, cognitive, and developmental, as well as neuroscience [15, 14, 16, 12]. Skills pertaining to software use were grouped hierarchically to reflect various psychological theories. Examples of skills include the ability to take initiative to act, the ability to transfer knowledge learned from one domain to another, the ability to be aware of one's own emotions and control their expression. The specific skills involved and the application of this hierarchical model is discussed further in

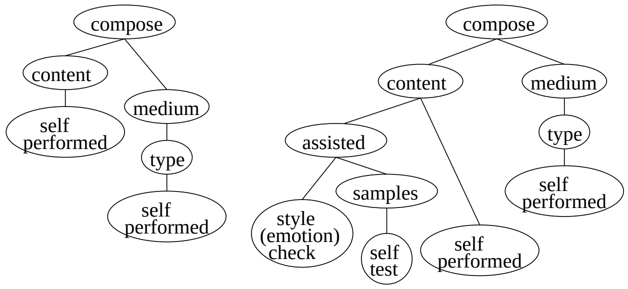

Figure 5. Alternatives: A (Left) and B (Right)

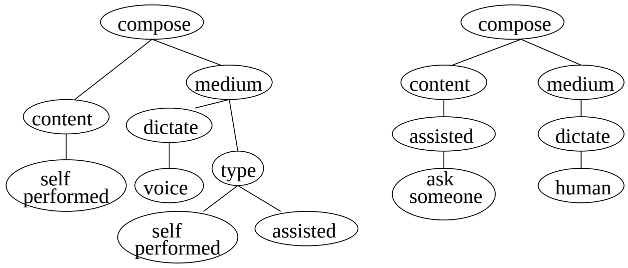

Figure 6. Alternatives: C (Left) and D (Right)

General:	**Required Skills**
all, except `ask someone`, `human`	selection, attention, memory
all `content` functions	initiation
all `content` functions except `self-performed`	self awareness
`Content` Branch:	**Required Skills**
style (emotion) check	evaluation of options
topic suggestion	evaluation of options
read samples	sentence processing, vision
ask someone	*none*
self-performed	composition
`Medium` Branch:	**Required Skills**
handwriting	character writing, vision
voice-to-text with edit	speech production, vision
voice	speech production
human	*none*
assisted	evaluation of options
self-performed	motor, vision

Table 1. Required Skills and Functions

[9].

With respect to the goal analysis graph, we extract the leaf tasks and associate skills to them. Table 1 shows the list of skills necessary to accomplish the tasks under the `compose` goal. The listed skills abstract away from implementation details and focus instead on requirements. The level of abstraction we adopt here corresponds to those in another study reported on potential TBI users [4]. In particular, `selection` skill requires motor input, which can be implemented in various ways, such as a mouse click or a key press. Similarly, the skill for various input `media` include traditional keyboard typing (requires motor skills), voice dictation (requires speech articulation skills), or voice recognition software with an editing option (requires speech articulation and motor skills).

Required skills refer to the skills that are necessary in order to perform a task. Table 2 shows the skills that are *supported* by a function. For example, a user may lack social judgment but choosing any of the several content functions will help this user develop the message content.

Combining the set of required and supported skills, we derive the alphabetic list of skill variables in our user profiles:

1. attention – cognitive ability to focus on one thing for a period of time
2. composition – language and creativity skill for writing sentences and short paragraphs
3. creativity – cognitive skill associated with generating ideas
4. evaluation of options – decision making when provided with several possibilities
5. initiation – taking a corresponding action when there is the desire to do something
6. character writing – language and motor skill
7. memory – short and long term ability to remember things
8. motor – includes typing, hand-eye coordination
9. selection – motor skill for coordinating the hand and eye when choosing the function
10. self awareness – social skill in recognizing that s/he needs assistance
11. semantic transfer – ability to transfer knowledge across domains
12. sentence processing – language skill to understand sentences

Functions:	Supported Skills
all `content` functions	creativity
all `content` functions except `self-performed`, `topic suggestion`	social judgment
self-test samples	semantic transfer
all `medium` functions except `self-performed`, `handwriting`	motor
handwriting	speech production

Table 2. Supported Skills and Functions

13. social judgment – appropriateness of language usage and interaction protocol
14. speech production – physical ability of vocal cords
15. vision – skill to recognize and produce images

Thus, these skills characterize alternatives and serve as an important parameter in selecting among them. For a particular alternative, each skill is marked as either "supports" (value of 1), "neutral" (0), or "requires" (-1)[1]. For example, alternative A (see Figure 5) has both the content and medium being `self-performed`. According to Table 1, the set of skills that are required by the `self-performed` leaf node for `content` are selection, attention, memory, initiation, and composition. From the same table, the set of skills that are required by the `self-performed` leaf node for `medium` are selection, attention, memory, motor, and vision. Therefore, each of these corresponding skill variables have a value of -1. Note that skills required by multiple leaf nodes still have -1 because the value denotes simply whether a skill is needed, not the degree to which it is needed. The same procedure is applied for supported skills in Table 2. We find that creativity is supported so that skill variable has a value of 1. The vector representation of alternative A matched with 15 skill variables is the following:

$$\mathbf{A} : [-1, -1, 1, 0, -1, 0, -1, -1, -1, 0, 0, 0, 0, 0, -1]$$

Summing up the values across the vector yields a total score of -6 for alternative A. Comparable values for alternatives B, C, and D can be calculated in an analogous way.

These are the skills that we need to assess for potential users in order to create the skills component of our user profile. Each person is grossly assessed with respect to these skills, and each skill is marked as either "strong" (value of 1), "neutral" (0), or "weak" (-1). Consider the vectors below, with u_1 representing a typical university student who has no problems with any of the email skills, u_2 representing an elderly user who is weak with memory and cognitive skills pertaining to fine details, and u_3 representing a patient with cognitive deficits who has problems with most email skills.

$$\mathbf{u_1} : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$$

$$\mathbf{u_2} : [-1, 1, 1, 1, 1, 1, -1, -1, -1, 0, 0, 1, 0, -1, -1]$$

$$\mathbf{u_3} : [-1, -1, -1, -1, 0, 1, -1, 0, -1, 1, -1, 0, -1, 1, -1]$$

These vectors form the skills component of a user profile.

2.3. Personal Preferences

The third component of our framework concerns user preferences. During the elicitation of user goals, target users identify not only important goals, but also how these goals are to be achieved, why they are important, and what scenarios make them important. The result of this elicitation process is a set of preferences. In the goal analysis framework, preferences can be naturally captured as *softgoals* – ill-defined goals that serve as criteria for evaluating alternatives by identifying positive or negative contributions from certain leaf goals. Consider Figure 7, an extension of the goal analysis on `compose`, with added softgoals (represented as clouds) and contributions made by leaf goals. We use `+` or `-` to denote whether the contribution is positive or negative respectively. The softgoals `improve social judgment` and `write interestingly` receive positive contributions from several content tasks. On the other hand, `independence` gets positive contributions from the tasks that are carried out by the user alone, and negative ones from tasks that are assigned to an external agent. Absence of a link between a task and a softgoal implies that the task does not have a direct influence to the respective softgoal.

Let p_1 be `improve social judgment`, p_2 be `write interestingly`, and p_3 be `independence`. Table 3 shows the preference scores according to the contributions depicted in Figure 7. A value of 1 is added for each positive contribution to a preference, 0 to each unmarked preference, and -1 for each negative contribution to a preference.

Taking the overall skill and preference scores for the alternatives, we get the following results: A has -4, B has 0, C -4, and D -1. Note that alternatives A and C

[1] Currently, skill levels range in [1,-1] for simplicity. The framework can support a wider skill level assessment by increasing the range of values, e.g., [5,-5].

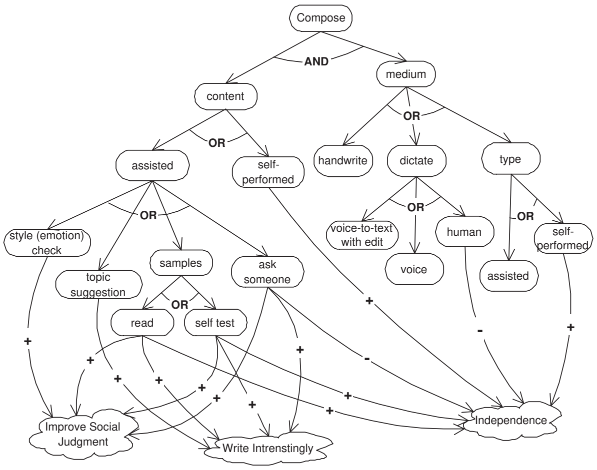

Figure 7. Softgoal Contributions

	p_1	p_2	p_3
A	0	0	2
B	2	1	3
C	0	0	2
D	1	1	-1

Table 3. Alternatives to Preferences

require similar skills and have similar preference contributions, while alternatives B and C require different skills but the preferences balance them out.

We also extend user profiles with preferences by treating them as an addendum to the profile vectors we have developed so far. The score assignment is similar to those of skills, with "preferred" corresponding to a value of 1, "don't care" to a 0, and "disapproved" to a -1. A regular user, u_1, may want to continue having independence in software use and not care about improving social judgment or writing interestingly as long as the message gets across. An elderly user, u_2, may want to be independent as well as writing interestingly. Finally, a cognitively impaired user, u_3, may consider all of these preferences to be important. Table 4 reflects these settings.

	p_1	p_2	p_3
u_1	-1	-1	1
u_2	-1	1	1
u_3	1	1	1

Table 4. User Preference Profiles

2.4. Evaluating Alternatives

In Section 2.1, we applied goal analysis to define the space of alternatives for fulfilling a set of initial goals. In Section 2.2, we identified the skills to analyze design alternatives and developed an abstract skill profile for a user. Similarly, in Section 2.3, we extended the user profile with preferences. Here, we carry out a sample evaluation of alternatives in our on-going email example.

The general algorithm for evaluating the set of alternatives is given in Table 5. Here, we define the data structure, `matrixAlts`, as a table with alternatives as its row elements and all the skill and preference variables as its columns. We assume that an alternative consists of a set of leaf nodes gathered from the goal analysis tree. For simplicity, we also assume that `matrixAlts` stores all alternatives in a graph. Later, we will lift this assumption when we optimize the algorithm.

```
for each row alternative, A, in matrixAlts
{
  - compare A against each column in user profile
  - determine score of each cell
  - tally up total for A
}
```

Table 5. Naive Ranking Algorithm

The second and third step in the for loop of the naive algorithm is a simple table look-up and summation respectively. However, the first step that compares alternatives against user profiles needs to be fleshed out. Basically, we are comparing two vectors of integers, v_1 (an alternative) and v_2 (a user profile), and our objective is to minimize the distance between them. Elements in each vector correspond to skill and preference variables. In our comparison procedure, we add v_1 to v_2 when we are comparing skill variables and subtract v_2 from v_1 when comparing preference variables. The results are stored in v_{tmp}, and taking the sum of the absolute values across the elements of v_{tmp} gives a single metric, which is a similarity score between the two vectors.

Intuitively, this simple distance metric returns a number measuring the distance between two vectors. This similarity measure helps us determine a suitable match among alternatives, given a user profile. Recall that users tend to specialize in a subset of the functionality provided by today's software. Therefore, we believe it is better to provide a less "bloated" application by matching the user's skills to an appropriate set of functions rather than just giving them everything. Thus, a person with many strong skills would be better matched with an alternative that exercises more skills because an expert user is more likely to perform a task herself rather than get support elsewhere. In the same manner, a person with many strong preferences would be better matched with an alternative that has positive contributions toward those preferences, while an alternative with negative contributions toward certain preferences would not matter to someone with weak preferences over them.

Following our procedure, we take the sum of the absolute value of the elements in each of the alternative vectors to find their similarity score with user u_1. The resulting scores are: $A = 11$, $B = 16$, $C = 12$, and $D = 19$. Therefore, as predicted, alternative A is most suitable for a regular user, with $C > B > D$ being the ranking of the remaining alternatives. The same procedure ranks the alternatives $C > D > A > B$ in order of decreasing similarity for user u_2, and $D > B > A = C$ for user u_3. This evaluation procedure gives us a ranking over the set of design alternatives.

3. Design Architecture

This section describes how the alternatives identified during requirements analysis are mapped to a software architecture that serves as a starting point for design. The section also describes the process by which one can arrive at a particular customization of the system at run-time.

3.1. Goals and Classes

To bring our customized alternative to the design level, we introduce one class for every task identified in the Goals-Skills-Preferences model. Although one class may eventually implement multiple tasks and vice versa, we begin the design process with a one-to-one correspondence between design and requirements elements. Figure 8 shows a simplified task-class association for a subtree of Figure 4. Alternative A, for example, will choose a `StandardInput` object, along with a sequence of the appropriate control objects (`Type`, `Medium`, etc.).

This task-class association involves specifying parameters that initialize each class instance. Depending on alternative from which the tasks were selected, the corresponding classes may conflict in run-time issues such as synchronization or resource usage. Solutions from distributed systems give insights to these problems [1].

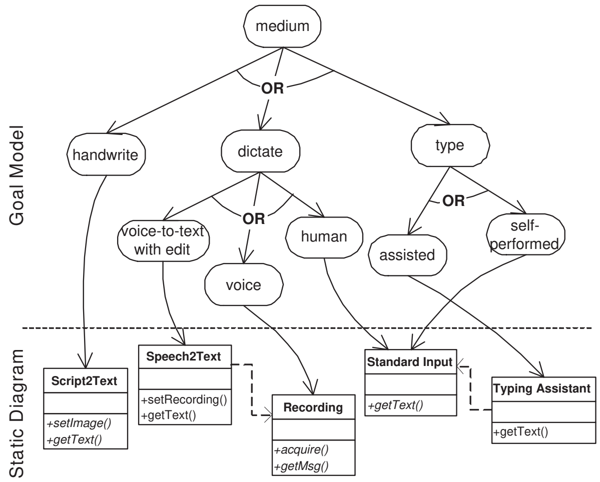

Figure 8. Task-Class Association for `medium`

After the task-class association, we further cast the internal nodes of the goal graph to classes that define elements of control flow (and name such classes *control classes*). The control structure of these classes is isomorphic to the structure of the goal analysis graph. In particular, classes corresponding to parent goals are responsible for calling the methods of one or more of the classes corresponding to goal children. Which methods are actually called depends on whether the corresponding parent goal is an AND or an OR one. If the goal parent has an AND relationship with its children, then its corresponding control class will call the methods of all its children classes, as shown in Table 6. On the other hand, a control class that corresponds to an OR goal will call one or more of its children based on some condition. Which child class method is actually called depends on customization parameters set at run-time. Note that other details about the design, such as the order in which child class methods will be called, will have to be introduced later in the design process.

The designer can, of course, revise this class structure, by combining several classes (e.g., merging an AND-class with its children) or by introducing others (e.g., interface classes).

3.2. Component Selection

Encoding customization information into the control classes can be carried out during two distinct stages:

Design Time Configuration. Given an elicited user profile, developers may select a set of classes that correspond to the best alternative with respect to this profile. We say that in this case, the user

```
class C_AND( class parameters )
{
  childA( parameters );
  childB( parameters );
}
class C_OR( class parameters )
{
  if( condition )
    childC( parameters );
  else if( condition )
    childD( parameters );
}
```

Table 6. Control Class Example Prototypes

commits to a specific customization at design time.

Run Time Configuration. While letting every component of the product be accessible to the user, the configuration process can be performed by her (or someone on her behalf) at run-time. The user may run a configuration procedure that asks her for profile information initially when she installs the software. Alternatively, the user is asked periodically if the user profile evolves over time. Web-based applications, for example, may follow this paradigm first by obtaining the user's profile and then by configuring the functionality that is suitable for her.

Note that the two options are not exclusive: at design time, a first subset of alternatives is selected, and at run-time, this subset is further filtered or re-selected. What is important to note is that the user does not deal with configuration at a technical level or decipher the options via an intermediary language, since our model exposes the user to only the choices that are relevant to the satisfaction of her goals. The act of customizing the software at run-time should not be a matter of selecting or re-selecting particular functions. Imagine a configuration wizard asking, "Would you like help from our typing assistant?", or showing a complicated dialog box, such as the mock-up in Figure 9. Instead, the user provides information from her profile, letting the configuration wizard automatically re-select the low-level functionality with respect to her goals.

4. A Case Study

In collaboration with clinicians and researchers at the University of Oregon, we applied our customization model to design a communication software for TBI

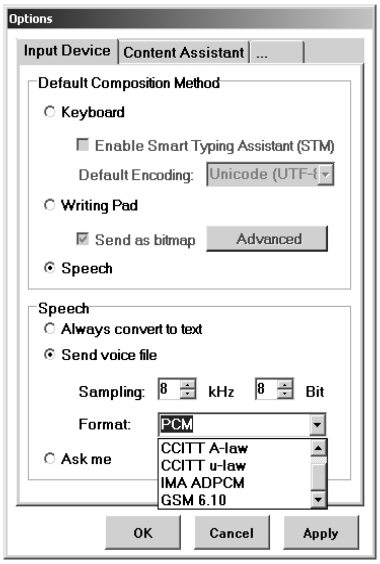

Figure 9. A Hypothetical Options Dialog Box

survivors. Because such survivors often suffer from social isolation, a communication software is an excellent medium for learning, while at the same time giving its users a sense of connecting with the rest of the world. In Fickas et al [4], focus groups were conducted to assess the needs (i.e., goals) of TBI survivors, along with reasons for choosing certain technologies. Although our discussion so far focuses on email, the feedback from the focus groups motivated us to choose a broader scope in developing customized software for TBI survivors. Instead of starting with a specific application (i.e., email) and then tailoring it to particular users based on periodic feedback, we began by identifying high-level goals for our users, applied our customization model, and identified alternatives for the software to be developed. Not only does this approach attempt to match user goals with a diverse set of skills from this cognitively impaired population, but the framework unearthed a huge customization space for exploration. Although TBI survivors may be an atypical example for target users, this population provides a strong argument for adopting systematic customization techniques into general software engineering practice. Our case study reports on an initial goal analysis for this population, based on the focus group feedback.

In our requirements model, the main stakeholder is a cognitively impaired user. Based on the description of assessed needs and technology preferences reported in [4], we came up with four top level goals: `Access Services` – the need to perform everyday tasks (e.g., get groceries delivered, banking), `Get Health Care` –

a special case of `Access Services` which refers specifically to remote inquiries or in-person visits to health professionals or centres, `Maintain Contact` – keeping in touch with friends, and finally, `Meet New People` – the user's desire to extend her social environment. Figure 10 shows a partial goal analysis graph for `Maintain Contact` to illustrate how it is decomposed.

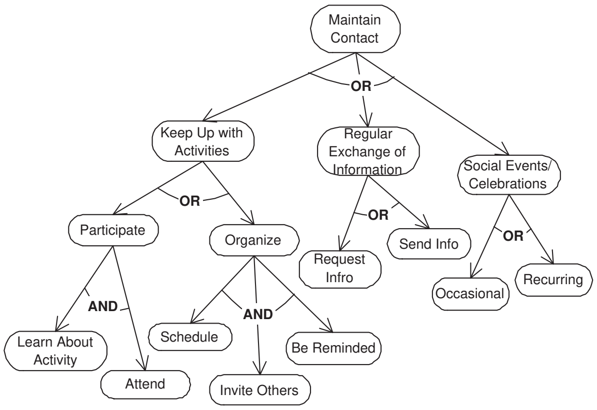

Figure 10. Partial Graph for `Maintain Contact`

Turning to preferences, we identified the following, most frequently occurring softgoals: `Avoid Disturbance of Other People`, indicating self-awareness: the user does not want to bother her friends; `Avoid Exposition of Impairment`, in order to prevent malicious people from taking advantage of TBI survivors by recognizing that they have an impairment; `Privacy`, whereby the content of any exchanged messages should be kept private to the communicators; and `Intellectual Stimulation`, i.e., the desire to carry out a challenging task and the satisfaction derived from achieving it. The skill variables of the target users are currently under development.

In terms of size, the current version of our goal analysis has about 750 nodes. A significant part of the goal analysis consists of common subtrees (e.g., options for input devices). The number of alternatives that the four top-level goals support are $4.5 \cdot 10^4$, $1.6 \cdot 10^8$, $4.9 \cdot 10^{11}$ and $3 \cdot 10^6$, in the order listed above. A tool was developed to analyze the graphs in the goal analysis. An attempt to enumerate 10^6 alternatives using a naive goal graph traversal algorithm (cf. Section 2.4) that stores k alternatives, for some small constant $1 < k < 20$, takes about 30 seconds on a Pentium IV, with 2.5Ghz CPU and 512M memory. In this case, the search for best alternatives is computationally manageable. On the other hand, the same algorithm ran on $4.9 \cdot 10^{11}$ alternatives hit the proverbial exponential explosion, and required several days to complete. Since the selection of most preferred alternatives is a key computational task in our framework, we are exploring heuristics that are computationally tractable and lead to near-optimal results.

5. Concluding Remarks

As software becomes omni-present, we need to develop new techniques for making it customizable for large classes of users, with diverse needs, skills, and preferences. These techniques need to support customization at a much finer-grain than is currently possible. Software that assists people to live their lives despite disabilities constitutes an important class of applications where such high degree of customization is essential. This paper proposes a framework that supports the design of highly customizable software by adopting goal-oriented analysis techniques. The workings of our framework have been illustrated with a simple example involving the design of a customizable email system. We are currently applying it to a case study involving TBI patients. We are also exploring effective heuristic techniques that generate best or near-best alternatives for achieving top-level goals for a particular set of skills and preferences. The novelty of our framework rests on the fact that it casts software customization as a requirements analysis problem, compared to other approaches that study it as a design problem. However, our proposal is definitely work-in-progress.

The most relevant work to our proposal is [6], where the use case notation is extended to represent different types of software variability from a requirements perspective. The key difference between this work and ours is in the choice of goals vs. use cases as fundamental concepts for representing and analyzing variability. With goals, one can explore alternatives even before there is a system to discuss (e.g., the goal `Maintain Contact`). With use cases, on the other hand, alternatives are explored in the context of system use. Thus, both proposals treat software variability during the requirements phase. However, our proposal is focusing primarily on earlier stages of requirements analysis than the use case one. Another point of difference is that goals offer a clear conceptual framework for defining variability space (through AND/OR decompositions). The situation is less clear with use cases, which sometimes represent goals and sometimes tasks [5].

Apart from dealing with the complexity of our alternatives selection algorithm, we are also looking at a number of other extensions of the framework. In particular, we are studying how multiple stakeholders may be handled in our work. Goal analysis in i* [17] supports goal analysis for multiple stakeholders. In this setting, goals can be delegated by one stakeholder to another, leading to networks of strategic *dependencies*. This extension will require novel conflict resolution mechanisms for situations where various goals, skills, and preferences among relevant stakeholders are mismatched. For example, a TBI survivor may be very enthusiastic about emailing a particular friend, so he sends several messages every day for several weeks. The friend is initially cooperative, but she is eventually overwhelmed and stops replying to his messages temporarily. In reaction, he sends even more messages, thereby overloading her mailbox and aggravating her. To prevent stakeholders from falling into such traps, the system should monitor the interaction between email partners to make sure they remain within accepted social norms. Many interesting research questions arise in recognizing and dealing with such issues. Moreover, we are working on modeling changes to a user profile at run time. Clearly, any set of personal profile parameters may change over time. But how do we detect these and how do we update the customization of the running system? Run time customization, or *adaptation*, is an open problem with little work in the literature. How should changes be made? Do we require that users perform periodic “checkups” where they are prescribed software updates? Clearly, much remains to be done.

6. Acknowledgments

We are grateful to Stephen Fickas (University of Oregon) and other members of the CORE team for sharing with us data on their on-going project. We are particularly grateful to Stephen for his encouragement (and nagging) throughout. We'd also like to thank the anonymous reviewers for giving us detailed and constructive feedback on an earlier draft of the paper. This research was partially funded by the Natural Sciences and Engineering Research Council (NSERC) of Canada and the Bell University Laboratories.

References

[1] G. Andrews. *Foundations of Multithreaded, Parallel, and Distributed Programming.* Addison Wesley, 1999.

[2] R. Baecker, W. Buxton, J. Grudin, and S. Greenberg, editors. *Readings in Human-Computer Interaction: Toward the Year 2000.* Morgan Kaufmann, 1995.

[3] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *The Science of Computer Programming*, 20, 1993.

[4] S. Fickas, L. Ehlhardt, M. Sohlberg, and B. Todis. Personal requirements engineering. Technical Report 45-02, Computer Science Department, University of Oregon, Eugene, USA, 2002.

[5] M. Fowler and K. Scott. *UML Distilled.* Addison Wesley, 1997.

[6] G. Halmans and K. Pohl. Communicating the variability of a software-product family to customers. *Software and Systems Modeling*, 1, 2003.

[7] E. Horvitz. Principles of mixed-initiative user interfaces. In *CHI*, pages 159–166, 1999.

[8] 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.

[9] B. Hui. *A Taxonomy of Software Skills and Its Applications.* Department of Psychology, Wilfrid Laurier University, November 2002.

[10] K. Kang, S. Kim, J. Lee, and K. Lee. Feature-oriented engineering of pbx software for adaptability and reuseability. *Software Practice and Experience*, 29(10):875–896, 1999.

[11] 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.

[12] M.-M. Mesulam. *The Principles and Behaviour of Cognitive Neurology.* Oxford University Press, 2000.

[13] J. Mylopoulos. *Conceptual Modeling and Telos.* In P. Loucopoulos and R. Zicari (eds). Conceptual Modeling, databases and CASE: An Integrated View of Information System Development. Wiley, 1992.

[14] S. Reed. *Cognition: Theory and applications (5^{th} ed).* Wadsworth, 2000.

[15] D. Shaffer. *Social and Personality Developmental.* Brooks/Cole, 1999.

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

[17] E. Yu. *Modeling Strategic Relationships for Process Reengineering.* PhD thesis, Department of Computer Science, University of Toronto, Toronto, Canada, 1995.