Configuring common personal software: a requirements-driven approach

Configuring Common Personal Software: a Requirements-Driven Approach

Sotirios Liaskos Alexei Lapouchnian Yiqiao Wang Yijun Yu Steve Easterbrook
Department of Computer Science, University of Toronto
10 King's College Road, M5S 3G4, ON, Canada,
{liaskos, alexei, yw, yijun, sme}@cs.toronto.edu

Abstract

We investigate the personalization capabilities of common personal software systems. We use a typical e-mail client as an example of such a system, and examine the configuration screens it offers to its users. We discover that each configuration value reflects each of the ways with which the user goals can be satisfied. Thus, we construct a goal model in which alternative ways for satisfying high level goals are matched with alternative system configurations. This way, automatic configuration of the system by reasoning about the overlaying goal model can be achieved. We find that the vast majority of the configuration options that refer to system functionality can be configured using this method, facilitating thereby the personalization tasks for users with no technical background, and ensuring, at the same time, consistency and meaningfulness in the configuration result.

1. Introduction

With the term *common personal software*, one can refer to software systems that are developed to be installed and used in home or mobile personal computing and communication devices, in order to support everyday life activities such as personal communication, learning, entertainment etc. For example, one can use a word processor to write one's school homework, a web browser to read the news, an e-mail client and several instant messaging systems to stay in touch with one's friends and relatives or media viewers and editors to maintain one's photograph/video albums. Although such systems are used extensively at the workplace as tools that support business productivity, they are also intended to be used as personal software systems.

When they are viewed as such, though, two basic issues may arise that make the corresponding requirements engineering effort difficult. Firstly, unlike software systems that are used at work, personal software cannot be seen as a production tool to which the user has to adapt as part of her responsibilities. On the contrary, because personal software generally faces the unrestricted human creativity and behavior in their leisure hours, it is the designer's responsibility to make the system adapt to the goals, preferences and abilities of the user. Secondly, the intended users of personal software are many and diverse to such a degree, that it is practically impossible to acquire requirements and produce a unique software system for each one of them. Hence, there is a need to develop software that can be *easily personalized* to accommodate the needs of each intended user as well as possible, without requiring from them any effort or technical knowledge.

However, a close look at today's practices reveals that ease of personalization is still questionable. In the general case, the user purchases/downloads one of the system's "editions" (e.g. the "professional" versus the "home" edition), installs the components that she needs and further configures the details of the installed system hoping to perfectly tailor it to her needs and abilities. In all these personalization phases, the user deals with a number of screens that display features (or packages of features) to be selected, deselected or adjusted through the definition of parameters. In some cases, special configuration agents ("wizards") are used to direct the user configure the necessary groups of options. But the parameters and options usually refer to the solution domain rather than the domain of the problems they can potentially solve. Hence, on one hand, they are expressed in a technical language that is not understood by users with little or no computer expertise. On the other hand, even if the user understands what she configures, she is unaided in her effort to come up with the configuration that best suits her needs. Further, the number of configuration options inevitably increases as the functionality of systems grows, making personalization an increasingly cumbersome process.

We believe that the basic problem behind the current practice is that it does not take into account the *user goals* behind configuration. In other words, the designers of the configuration widgets do not seem to ask *why* the particular set of options should be exposed to the user, and *why* a user

would configure them this way or another.

In this paper, we address this problem and propose a *goal-oriented* method for understanding, reorganizing and levering the configuration options of personal software. First, we develop a goal model on top of the software configuration options that acts as a mediator between them and the user. Then, through qualitative analysis of the goal structure, we are able to automatically translate the user's high level goals and preferences into configurations that satisfy these goals.

To illustrate our approach, we consider configuration that takes place after a personal software system that has already been installed on a computer. At this point of the life-cycle, configuration is done through the use of specially designed dialogue windows under the title "Options", "Preferences", "Customize", "Settings" etc (from now on: *the "Options" widget*). The vast majority of software of the genre contains at least one such dialog window. In the rest of the paper, we will use the term *configuration*[1] to refer to personalization that takes place at this phase. We focus on the "Options" widget of a particular e-mail client, namely Mozilla Thunderbird 0.5 ([12]). Mozilla Thunderbird is the e-mail component of the Mozilla application suite, offered as a separate stand-alone product. We chose Mozilla for this study because it is the most popular open-source and multi-platform e-mail client available to date. In light of Mozilla's accessibility, no barriers are introduced in replicating, confirming or challenging the findings presented in this paper. We furthermore believe that observations made in Mozilla reflect to an adequate degree the general practice for designing "Options" widgets.

Thus, in Section 2.1 we take a close look at the "Options" widget of Mozilla Thunderbird. We examine its structure and discuss its problems. Then, in Section 2.2, we discuss related research work. In Section 3, we present a series of steps that can be followed for the construction of the goal model, and, in Section 4, we show how we can use the goal model to configure the software system. In Section 5, we also investigate the use of goal model parameters that could facilitate the association of goal models with low level parameter configurations. We discuss how well our method performed in practice in Section 6 and we conclude in Section 7.

2. Current state of Research and Practice

2.1. Exploring the "Options" Widget

Let us have a close look at the "Options" widget of Mozilla Thunderbird. We focus on the screens under `Tools -> Account Settings...` and `Tools -> Options...` menu items. The former contains default settings for each of the accounts that have been created for this instance of the application. The latter contains application-wide default settings.

We define *configuration item* to be any visual control (e.g. text boxes, drop-down menus, sets of radio buttons) with which the user can view and change configuration information. The *domain* of the configuration item is the set of possible *values* it can take. For instance, the text box `Account Name` accepts a (practically) infinite number of strings, whereas the check-box labeled `Check spelling before sending` accepts only two values ("enabled" or "disabled"). A *configuration* of a set of items is simply a set of values to which these items can be set.

We examine a total of 17 configuration screens that are included in these two dialog windows, ignoring a small set of dialogs that can be called from these screens. We count a total of 120 configuration items. Although we cannot assume that this is the complete set of items that the system offers, they certainly constitute a good sample for discussing the nature of the problem and for performing an initial evaluation of our proposal.

We first investigate what entity each configuration item configures. We observe that there are five such entities: the system, the application, the e-mail account, the outgoing message and the incoming message. Each configuration item configures one of these entities. But it may be defined in a different entity from the one it actually configures. For example, although the boolean item `Check spelling before sending` configures each outgoing message, the "Options" widget keeps the configuration value as part of the e-mail account. This value serves as the default value for every message send out from the particular account. 69 out of the 120 configuration items we examined would be part of objects other than the ones they actually configure.

This scheme (i.e. pushing configuration options to container entities as defaults) serves an obvious practical reason: we do not want the user to configure "from scratch" every object she creates. Nevertheless, the choice of accommodating these defaults to one of the entity's containers is arbitrary. For example, the default value for whether to `Check spelling before sending` could depend on who the recipients are or which time of the day it is send; these possibilities are not less reasonable than having the particular default option depend on the account from which the message is send.

More problems are revealed if we approach the configuration items from a point of view of their semantics. Figure 1, for example, depicts the screen where the user can configure the fonts to be used for messages. In short, even if the users know what the difference between "Monospace",

[1] The use of this term should not be confused with the one found in the literature about configuration management.

"Serif", "Sans-serif" and "Proportional" is, it is not certain whether they have a strong opinion about which of the available values is the one they really need.

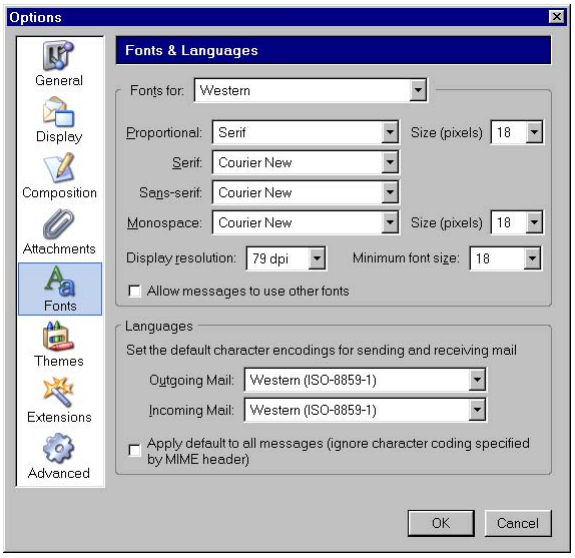


Figure 1. Configuring the Fonts for Mozilla Thunderbird

Furthermore, the number of different combinations of values in the upper frame of the screen depicted in Figure 1 is at least 80 billion. Detection of configurations that will not reasonably be used (obviously there are such) is arguably possible.

We can summarize the problems with the current practice of designing "Options" widgets as follows. Firstly, the configuration items are arbitrarily many and organized in an ad-hoc manner, making it impossible for the user to handle them without significant effort commitment. Secondly, due to this complexity, a one-size-fits-all approach to configuration has to be followed. Such an approach demands the configuration parameters of an object to be statically defined as default configurations of the entire class of such objects. Thirdly, the user might not be capable of understanding what the available options mean; even if she knows what the available options mean, the user might fail to understand which of them is the most appropriate according to her needs, capabilities and generic preferences. Fourthly, there might be configuration items or values, that do not interest a particular user, as well as combinations of configuration values that could not reasonably apply to any user.

In conclusion, the current approach to configuring common personal software is one that ignores the importance of *user requirements*. The users are expected to literally intervene to the software system's design, by interpreting their goals and requirements into configurations, without any external help. As we discuss in the next section, requirements for software customization is a subject that has not enjoyed the appropriate attention by the SE and RE communities.

2.2. Related Work

The problem of configuration, as described so far, has mostly been studied from a Human Computer Interaction point of view. In [6], a number of users are observed subject to when and how they customize a number of software applications. Among other things, it was found that users cannot afford the time needed to customize the software, given that there is also a risk of failing to achieve the intended result. Thus, users would not go to the trouble of configuring their systems, when this seemed "too hard" and the respective documentation was not rich enough. In a similar investigation, reported in [10], it was also found that users are more likely to engage to customizing their systems when the respective task is easy.

In [8], McGrenere et al. propose a method for customizing the user interface of a word processor. According to the proposed technique, the interface should initially contain only a basic core of the system's functionality. While using it, the user gradually adds the functions she needs, leading to an interface that best suites to her. However, the user is assumed to know how to use the system with optimal efficiency and effectiveness and is therefore capable of making the correct personalization choices easily. Furthermore, the focus remains on the availability of functions on the main interface, which is only a subset of what can be customized in a software system.

In [7], the construction of models that describe the design rationale of an intended user interface is proposed. The models imply a space of design alternatives, each contributing positively or negatively to high level selection criteria. Although the modeling idea is very similar to the one that we are using here, the design rationale paradigm alone, as described in [7] is not sufficient for our purposes. Firstly, it does not follow the necessary user-centered approach, as it assumes that the same rationale model can be read from both users and designers. Secondly, it does not necessarily focus on describing the intentional content of a potential customization decision; it rather focuses on articulating and understanding visual and behavioral aspects of the system without asking why these aspects are considered in the first place.

From a Requirements Engineering point of view, work on requirements *monitoring* ([2]) is also related to our discussion. In [1], Feather et al. have proposed the use of monitors that detect violations of requirements specifications, and change the systems' behavior accordingly. The goal is to optimize the effectiveness and efficiency of low level task performance according to observed behavior. This can be

done by changing the parameters of the system's functionality or even switching to a different design. Again, though, high level user preferences or system-wide qualities are not taken into account.

Nevertheless, goal models can effectively be used to represent such high-level concerns and their relationships ([9]). They can also serve as a bridge between these concerns and the low level functionality of the system: each elementary low level decision contributes to the satisfaction of global soft-goals to some degree. Conversely, the total contribution to soft-goals implies particular low level decisions ([5]). As we discuss in the next section, these ideas can effectively be used to simplify the task of configuring software.

3. Identifying the Configuration Goals

We now present the details of the proposed method. The method is based on examining the candidate values of each configuration item and identifying both the user goals that each selection achieves and the qualities it contributes to. The result is a goal model built on top of the configuration items. The user can then configure her system by reasoning about the resultant goal model.

The process of constructing the goal model is organized in a sequence of steps, which we describe in the following subsections, using examples from the Mozilla Thunderbird case.

3.1. Step I: From Configuration Items to Goals

We create a list of the configuration items of our "Options" widget. For each of the items we ask two questions:

- "What is the goal that is achieved by the function being configured?"
- "What high level aspects of the user experience are influenced by the value that is selected in this item?"

The first question refers to the functional goal that is associated with the particular configuration item, for example `[Use Encryption]` or `[Check Spelling]`. The second question refers to qualities of the system as a whole that reflect generic user goals and preferences. `[Privacy]` is an example of such a goal. These goals do not have a clear-cut criterion which we can use to decide whether they are satisfied or not. Thus, they are rather *satisficed* ([9]), that is satisfied to some degree and with some uncertainty.

We will refer to these two goal types as goals and soft-goals respectively, in accordance to the *i** ontology for building goal models ([13]).

Figure 2. Frequency of checking for new messages

Consider, for example, the pair of configuration items depicted in Figure 2. This pair of items allows the user to define whether she wishes the system to periodically connect to the mail server and check for new messages and if yes, how often this should happen.

At this step our method requires us to understand what goal we are trying to achieve by the function being configured. Trivially in this example, the purpose is to `[Automatically Check for New Messages]`. But different alternatives in terms of the frequency with which this check can occur, imply different contributions to particular soft-goals. Thus, having very frequent checks contributes positively to the soft-goal `[Increase Availability]`, in a sense that the user can respond to the senders promptly, making communication more intense and productive. But this contributes negatively to the goal `[Reduce Network Use/ Dependency]`, as a connection to the mail server will need to be established every little while. Further, the subsequent notification frequency contributes negatively to the soft-goal `[Reduce User Distractions]`. Obviously, when less frequent checking is chosen, the contributions are defined accordingly.

3.2. Step II: Alternative Options as OR-decompositions

Having identified the goal behind the function being configured, we continue by assuming that each value of the configuration item implies an alternative way to satisfy that goal (perhaps including the option not to satisfy it). By picking one of the possible values for the configuration item, the user specifies one of her possible intentions on how she expects the system to satisfy a goal of a higher level. This conceptual pattern can be nicely illustrated by an OR-decomposition of the original goal. Such decomposition is depicted in Figure 3.

The goal `[Automatically Check for New Messages]`, which was identified for the items of our example in the previous step, is decomposed into a small set of subgoals (ovals), each expressing the different frequencies with which the checking can be performed. Each of the alternative subgoals is associated to a specific value of the configuration items (boxes). Thus, `[Check as frequently as possible]` would check for

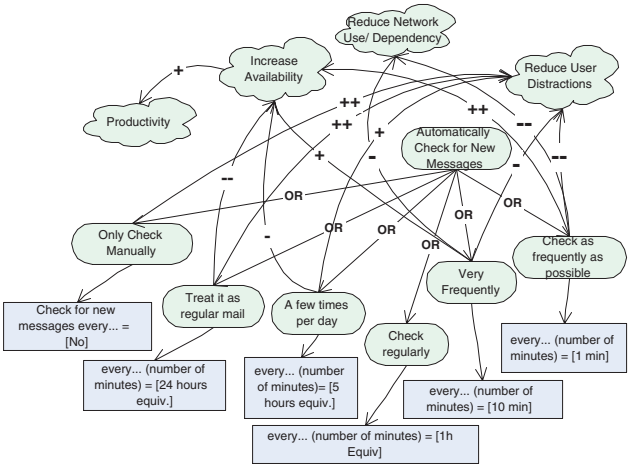


Figure 3. Valuations as Goal Attainment Alternatives

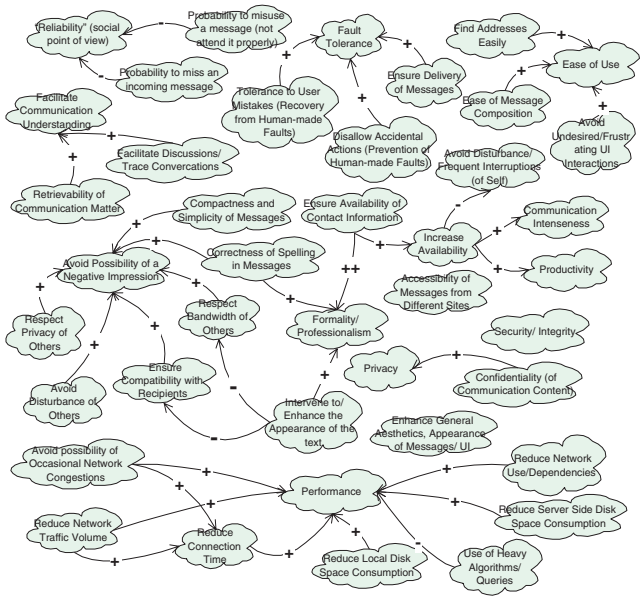


Figure 4. Soft-goal Analysis

new messages every minute, whereas `[Treat it as Regular Mail]` means that the system should check for new mail daily.

By expressing groups of configuration items as goal decompositions, we have reduced a practically infinite domain to a small set of meaningful values. Moreover, we have understood why the particular configuration items are there and what would alternative values of the item mean in satisfying (or not satisfying) this purpose.

3.3. Step III: Alternatives and their Impact to Softgoals

Having defined the OR-decomposition in the previous step we can now question each alternative subject to its impact on the soft-goals we identified in the first step. In Figure 3, soft-goals are represented using cloud shaped elements.

So, for example, choosing `[Treat as Regular Mail]`, which in turn implies that the checking occurs every 24 hours, contributes positively to the soft-goal `[Reduce User Distractions]`. But, as we saw previously, this choice also contributes negatively to the soft-goal `[Increase Availability]`. In Figure 3, we represent this by adding the corresponding *i** contribution links between the goals and by choosing the contribution degree to be in the range from -2 to +2. The contribution to these soft-goals can be further propagated to soft-goals of a higher level. In our example, the soft-goal `[Increase Availability]` contributes positively to a more generic one we have called `[Productivity]`.

3.4. Step IV: Integration

If we follow the same steps for every configuration item of the system under investigation, we end up with a forest of elementary goal decomposition trees. For Mozilla Thunderbird part of the result is depicted in Figure 8. A closer look at the diagrams will reveal that many soft-goals receive multiple positive or negative contributions from different goal trees. Moreover, soft-goals contribute to each other the way we described above. In Figure 4, we isolate all soft-goals in a separate diagram and relate them through contribution links, also introducing soft-goals of an even higher level, when possible.

The result of this process allows us to talk about the configuration of the whole system using a high level non-technical language. More interestingly, if we pose constraints on the total degree of contribution that particular soft-goals can accept, these constraints will be propagated to the low level, requiring particular configuration values to be set, without the user having to intervene.

In other words, as we discuss next, having concluded the development of the goal model, we can move conversely and use it in order to lever low level configuration.

4. Using the Goal Structures

4.1. Label Propagation

The models that have been developed in Figures 4 and 8 can be used for two types of diagrammatic reasoning. One

is the *forward label propagation algorithm* introduced by Giorgini et al. in [4]. The input to the algorithm is the degree by which each low level goal G_i is fully/partially satisficed or fully/partially denied (FS([G_i])/PS([G_i]) and FD([G_i])/PD([G_i]) respectively). The algorithm will then use the links between the goals to propagate the satisfaction (or denial) of the low level goals to the high level ones.

We can use this algorithm to understand the impact of an existing configuration to the high level user goals. Recall that each low level subgoal is related to a particular configuration of one or more items. Thus, given the configuration of these items, we may be able to match them exactly with one of the subgoals and label the latter as fully satisfied. Then, we can apply the algorithm to calculate the degree of satisficing (or denial) of each of the high level soft-goals. Of course, due to the sampling we have performed to reduce the size of the domains, many configurations might not have an exact correspondence to a low level goal. In this case, we select one or more goals that are close to this configuration and assign partial satisfaction to these goals. For instance, if for the configuration option `Check Messages every x Minutes`, `x = 10` is associated with the goal `[Check very frequently]` and `x = 60` with the goal `[Check regularly]`, if the actual configuration where `x = 30` the initial satisficing assignments would be PS(`[Check very frequently]`) and PS(`[Check regularly]`).

A more interesting use of the resultant goal graphs is the *backward label propagation* algorithm introduced by Sebastiani et al. in [11]. This algorithm accepts the desired degrees of satisficing (or denial) of a set of high level goals (the "targets") and searches for the values in the low level goals (the "inputs") that ensure these degrees. In other words, the user can define the degrees by which she requires a set of important soft-goals to be satisfied or denied and have the system configured in a way that these requirements are met. For example, if the user defines the set {PS(`[Security]`), FS(`[Communication Intensiveness]`), FD(`[Formality]`)}, the algorithm will select those low level goals that ensure these values (e.g. FS(`[Check very frequently]`), among others), which in turn implies specific values of the associated configuration items (e.g. `Check every 1 minutes`).

4.2. Redesigning the "Options" Widget

Bottom-up and top-down analysis of impact propagation introduces new possibilities for the design of configuration interfaces. Interfaces that support *goal-based configuration* can be built in order to function in two modes. The *untrusted* mode allows the user to edit low level details of the system, while ensuring she maintains awareness of the impact of their options (*"macro-awareness"*), using bottom up analysis of the goal model. The *trusted* mode, allows the user to set desired degrees of soft-goal satisficing while making her aware of the changes that automatically occur in the low level options (*"micro-awareness"*). Different kinds of users may choose different modes and appreciate micro- or macro-awareness in different degrees. Thus, computer experts and "techies" will probably choose to work with the untrusted mode, whereas, elderly, children and people without technical background may choose to use the trusted mode and even turn micro-awareness off.

The exact way with which desired satisficing of soft-goals is defined in the trusted mode poses an interesting interface design problem. The obvious practice of directly assigning degrees may prove unintuitive, especially if we consider the inevitable trade-offs between satisficing of different soft-goals.

A more promising possibility is to demand optimized contribution over a subset of soft-goals ranked subject to their priorities for a particular user. For example, the user might declare that a configuration that meets her needs must first ensure maximum `[Privacy]`, then maximum `[Performance]` and then maximum `[Ease of Use]`. Adapting the top-down label propagation algorithm to respond to such input is straightforward.

Further, the decrease in the number of parameters that need to be set to a small set of soft-goals allows for more flexible configuration of the system, as we can allow the default configuration of our system vary depending on several meaningful factors. For example, the user can easily declare variable degrees of `[Privacy]` for each part of the day or different degrees of `[Formality]` and `[Communication Intenseness]` for different senders/recipients, expecting that the system will dynamically configure itself accordingly.

5. Gaining Accuracy: Parametric Goal Models

The method we have been presenting relies significantly on the correct use of numbers. These occur in two cases:

1. When the domain of a configuration item that is continuous needs to be appropriately sampled in order to be associated to a small set of subgoals.

2. When a contribution to a soft-goal needs to be defined.

Consider, for instance, the configuration items given in Figure 5. With these items, the user can define whether the system should ask permission before downloading a message that exceeds a particular size. The corresponding goal decomposition can be seen in Figure 6 (ignore the soft-goals for the moment). The user may choose either to allow all messages to automatically be downloaded, or to allow those that do not exceed a particular size category (small

(i.e. without attachments), medium and large). However, we should not expect a general agreement among users of what can be considered as a medium message (or a small or a large one respectively).

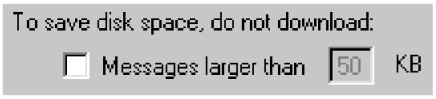


Figure 5. "When should I truncate?"

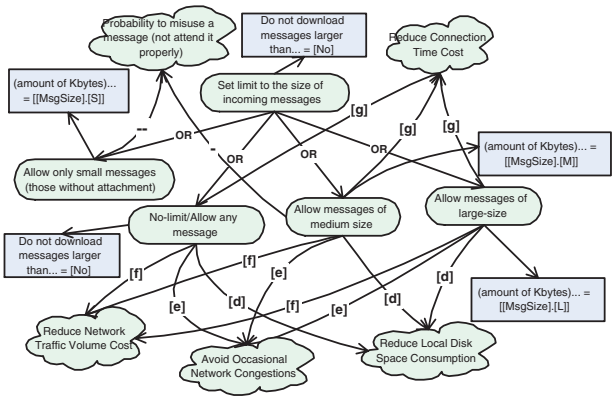


Figure 6. Parameterizing goal models

Hence, we should rely on parameters which are valuated using *facts* particular to each user. In our example, we can use simple statistics based on the user's e-mail traffic. The result would be descriptions like the following:

[MsgSize].[S]: `Average size of messages that do not contain attachment. 50k by default`

[MsgSize].[M]: `Average size of messages that contain at least one attachment. 1Mb by default`

...

Such parameter descriptions are essential part of the goal model as they further enhance its adaptability to different cases of users.

The same approach can be used to parameterize degrees of contribution. In Figure 6, let us focus on two of the softgoals that are influenced by the available options. The one is `[Avoid Occasional Network Congestions]` that may occur when downloading a large attachment through a slow connection; the goal refers to the frustration or reduction of productivity this may cause. The second is `[Reduce Connection Time Cost]` which might be of importance depending on the charging policy of the internet service provider.

The degree by which downloading a message of a particular size can hurt these goals cannot be generally assessed. Thus we will again use parameters. For the first goal we will base the estimation on the time it takes for the message to download given the available bandwidth:

$$\texttt{e} = \begin{cases} 0, & \text{if } \dfrac{\text{(selected msg size)}}{\text{(bandwidth)}} < 2 \text{ sec} \\ -1, & \text{if } 2 \text{ sec} < \dfrac{\text{(selected msg size)}}{\text{(bandwidth)}} < 1 \text{ min} \\ -2, & \text{if } 1 \text{ min} < \dfrac{\text{(selected msg size)}}{\text{(bandwidth)}} \end{cases}$$

Similarly, the contribution to the goal `[Reduce Connection Time Cost]`

$$\texttt{f} = \frac{\text{(selected msg size)}}{\text{(bandwidth)}} \times \text{(cost per second)} \times \text{(scaling factor)}$$

Here, the scaling factor expresses the "significance" of the objective cost for the particular user, in a way that the formula returns a value between -2 and 0. Models for parameters `d` and `g` can be constructed similarly.

Of course, all models we mention above are naive examples and are given in order to illustrate how we can define parameters in goal models. How elaborate the parameter models should be, varies depending on the degree of granularity and accuracy that needs to be achieved, the available domain expertise, as well as the cost for the acquisition of the appropriate measurements. In the worst case, a parameter is "hardcoded" with a fixed value. In a better case, state-of-the-art machine learning and monitoring frameworks can be used to model the goal parameters and acquire the necessary measurements respectively.

Parameterized goal models can provide a framework for achieving synergy between *adaptability* and *adaptiveness* ([3]). On one hand, the selection of a particular alternative in the goal model is a result of having the user to specify her intentions. On the other hand, parameter models reflect facts about the system, the context or even the user (e.g. patterns/models of her behavior), that allow the system to automatically adjust how exactly the user intention should be translated into the actual system configuration. Thus, if parameters are used, a system configuration may change not only because the user intentions changed, but also because the system decided to interpret them in a different way.

6. In Practice

In this section, we discuss our experience in applying the method to Mozilla Thunderbird.

A preliminary step is to understand what aspect of the system each item actually configures. In Table 1, we attempt a categorization: although the majority of the items configure *functional and behavioral* aspects of the system, many of them are mainly either what we call *structural data* (e.g. `Login Name`, `Server Address`, `Folder`

Aspect	#items
System	2
Funcion/Behavior	64
Input/Output Data Format/ Appearance	8
Interface Appearance	19
Structural Data	27

Table 1. Configuration items per configuration aspect

`Name`) or *user interface appearance* concerns (e.g. `Background/Foreground Color`).

Further, we try to identify which of the configuration items can be associated with user goals as advised in Steps I and II of our method. It turns out that there are four major categories of items according to their suitability to the construction of the goal models.

Suitable. These are configuration items that can be associated with a goal that is served by the functionality being configured. Moreover, goal decomposition that reflects alternative customization values is possible and each alternative has a different impact on generic properties of the system. Thus our method applies well in these items.

Non-Intentional. These are items for which different values do not reflect different user intentions or preferences. They rather refer to "objective" information about the user and the system. An example is the e-mail address of the user. Our goal oriented method could not apply to such configuration items.

Non-Trivial. These are items that should relate to some user goals, but neither these goals nor the subgoals to which they are decomposed nor the impact to any generic properties can be trivially articulated. An example is the panel orientation in Thunderbird's main screen. The user can select among three different orientation options. However, neither the intentions that lead to the selection of one of the three options nor how this influences general system qualities or generic user preferences can be expressed in a useful way.

Unnecessary. These are mainly configuration items for which it seems that only one value is meaningful; the purpose of the item is therefore challenged. For instance, the question whether the setting for the encoding of the incoming message should override the corresponding default setting (see Figure 1), should have a positive answer at all times; we could not find a case where the default settings of the client should have priority.

Notice that the result of the classification will depend on the subject that performs it. The authors attempted to classify the 120 items of this study into one of the above categories. If we combine the result with the categorization we produced in Table 1, some interesting observations can be made. Figure 7 illustrates the degree of suitability of our method with respect to the type of the items that are contained in the "Options" widget.

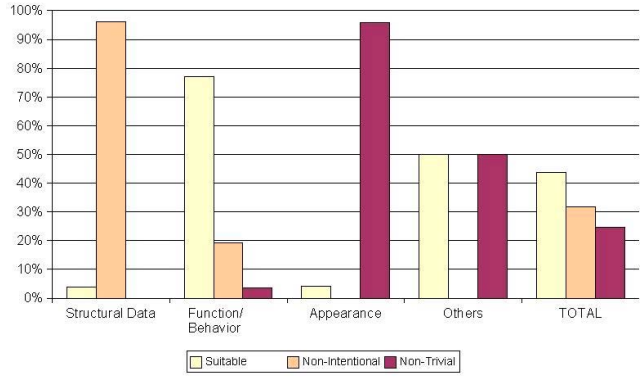


Figure 7. Coverage per Configuration Types

It is clear that the goal-oriented method can be applied when the configuration items are about functional and behavioral aspects of the system. However, items that constitute "structural data" cannot be associated with user intentions and cannot thus be approached using a goal-oriented method.

Interestingly, we would also fail to articulate goal descriptions for options that reflect the appearance of the user interface. However, there is no evidence that such options do not serve practical goals. For example, the selection of Serif over Sans-Serif fonts is known to facilitate ease of reading given that the font size and the resolution is adequate; Sans-serif fonts on the other hand are often characterized as "modern". But such rules are not necessarily agreed among users, unless empirical studies have indicated so (and experts are available to confirm it). Furthermore, when such practical considerations meet the aesthetics of the users, the result is not predictable; one could claim that the interface appearance should be something that the user should better customize directly.

In Figure 8, 32 of the configuration items for Mozilla Thunderbird are analyzed through our method. Through the soft-goal analysis of Step IV (Figure 4) the soft-goals that can be used to configure the respective items are as many as 12. These are the soft-goals that do not contribute to any goals of higher level (i.e. they are "sinks" in the goal graph) and we can use them in the procedures described in Section 4. Observe that if we introduced additional functionality to Mozilla, this would necessarily increase the number of re-

quired configuration items. However, it is unlikely that the number of soft-goals would be influenced by this increment.

7. Conclusions and Future Work

The motivation behind the work we presented in this paper is the improvement of the configuration practices that are followed in today's desktop applications. The central idea of our approach is that we can configure the numerous details of the system by only dealing with high level system-wide user goals. We presented a method for mapping these goals to the low level configuration options. Given this mapping, the users only need to communicate their high level strategy of the use of the system; the configuration details will be automatically set to support this strategy. We believe that this will certainly benefit the vast majority of the users who don't have (and don't want to have) technical knowledge required for configuring software following today's practices.

Thus, the usefulness of the method is clear when the configuration options refer to the core functionality of the software system. However, the relationship between goals and options that relate to pure user interface appearance issues has to be further investigated.

Furthermore, it would be interesting to see how the software vendors would react to such a possibility of organizing configuration items. One could hypothesize that thinking about configuration in a goal-oriented fashion would at least force analysts and designers reconsider the necessity of many existing configuration items. But, on the other hand, the effect of goal analysis to a requirements process is the achievement of a better score in terms of completeness: it may allow analysts think of important options that need to be added to the existing ones.

In terms of empirical research, further evaluation of the method needs to be done. A field study could show whether the users indeed observe their high level preferences and strategies being properly implemented. Moreover, could groups of users contribute to the construction of an even more accurate and realistic goal model and how?

Finally, how wide could the scope of a single soft-goal model be? Privacy, for instance, is not a concern about e-mail clients only; how could one use a soft-goal model to configure a set of separate systems that serve the same general purpose (e.g. communication in our case)? Certainly such an investigation would be of great practical interest.

Acknowledgments. We would like to thank John Mylopoulos for his precious feedback and overall support on this work, Ravin Balakrishnan for his valuable advice on user interface design aspects of our method as well as Mehrdad Sabetzadeh for his very helpful comments on earlier drafts of this paper. Funding for this work was provided by the Precarn/IRIS project titled "Intelligent system requirements for cognitively impaired individuals", the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Bell University Labs (BUL).

References

[1] M. S. Feather, S. Fickas, A. Van Lamsweerde, and C. Ponsard. Reconciling system requirements and runtime behavior. In *Proceedings of the 9th International Workshop on Software Specification and Design*. IEEE Computer Society, 1998.

[2] S. Fickas and M. Feather. Requirements monitoring in dynamic environments. In *Second IEEE International Symposium on Requirements Engineering*, pages 140–147, York, England, March 27 - 29 1995.

[3] G. Fischer. Shared knowledge in cooperative problem-solving systems - integrating adaptive and adaptable components. *Adaptive User Interfaces*, pages 49–68, 1993.

[4] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Reasoning with goal models. In *Proceedings 21st International Conference on Conceptual Modeling ER 2002*, Tampere, Finland, October 7-11 2002.

[5] B. Hui, S. Liaskos, and J. Mylopoulos. Requirements analysis for customizable software: A goals-skills-preferences framework. In *11th IEEE International Requirements Engineering Conference*, 2003.

[6] W. E. Mackay. Triggers and barriers to customizing software. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 153–160. ACM Press, 1991.

[7] A. MacLean, R. M. Young, and T. P. Moran. Design rationale: the argument behind the artifact. *SIGCHI Bull.*, 20(SI):247–252, 1989.

[8] J. McGrenere, R. M. Baecker, and K. S. Booth. An evaluation of a multiple interface design solution for bloated software. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 164–170. ACM Press, 2002.

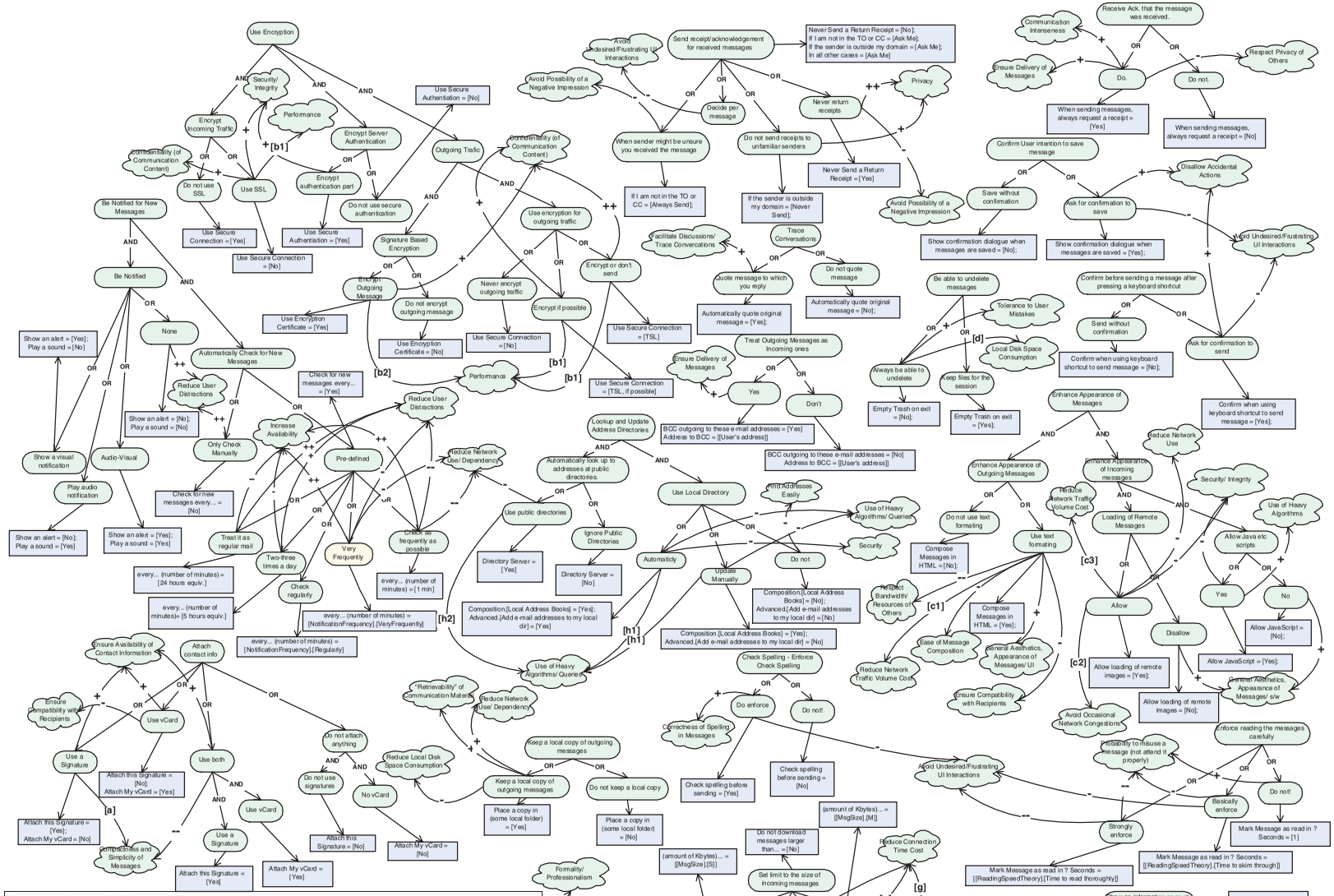
[9] J. Mylopoulos, L. Chung, and B. A. Nixon. Representing and using nonfunctional requirements: A process-oriented approach. *Software Engineering*, 18(6):483–497, 1992.

[10] S. R. Page, T. J. Johnsgard, U. Albert, and C. D. Allen. User customization of a word processor. In *CHI '96: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 340–346. ACM Press, 1996.

[11] R. Sebastiani, P. Giorgini, and J. Mylopoulos. Simple and minimum-cost satisfiability for goal models. In *16th International Conference on Advanced Information Systems Engineering*, Riga, Latvia, June 7 - 11 2004.

[12] Website for Mozilla Thunderbird e-mail system. http://www.mozilla.org/products/thunderbird/.

[13] E. S. K. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE Int. Symposium on Requirements Engineering (RE'97)*, Washington D.C., USA, January 1997.



CONFIGURATION PARAMETERS GUIDELINES:
[ReadingSpeedTheory].[TimeToReadThroroughly]: Acquire with a test. 30 sec if not possible.
[ReadingSpeedTheory].[TimeToSkimThrough]: Acquire with a test. 5 sec if not possible.
[ServerQuotaTheory].[MaxTimeWindow]: The maximum allowed usage of server space over the average volume of messaging traffic per day.
1 year if there is no limit.
[NotificationFrequency].[VeryFrequently]: So that at most 1 message arrives each time (based on estimated rate). 5 mins if not available.
[NotificationFrequency].[Regularly]: So that at most 3 messages arrive each time (based on estimated rate). 60 mins if not available

CONTRIBUTION PARAMETER GUIDELINES:
[a]: Size of the signature compared to the expected size of the message
[b1]: Performance of SSL and TSL versus the non-encrypted version - [b2]: Time for the encryption algorithm (0 if only a few seconds)
[c1]: Bandwidth (0 if 56kbps or more) [c2]: Depends on time it takes for the image to download (image size estimation required). [c3]: Depends on estimated size of the image and cost per MB (0 if cost is not per MB)
[d]: Available Local Disc Space versus inbox accumulation rate (0 if disc space affords several years of e-mailing)
[e]: Depends on time it takes for the message to download (requires message size estimation)
[f]: Depends on estimated size of the message and cost per MB (0 if cost is not per MB)
[g]: Depends on time it takes for the message to download (message size estimation required) and cost per second
[h1]: Look-up address local book performance; [h2]: Look-up remote book performance
[i]: Depends on the time window selected, the server quota, the rate and size of the incoming e-mail traffic.

Figure 8. Configuration Goals for Mozilla Thunderbird