A Language-Based Approach to Variability Analysis

Bruno Santana da Silva, Simone Diniz Junqueira Barbosa, Julio Cesar Sampaio do Prado Leite
Departamento de Informática, PUC-Rio
R. Marquês de São Vicente, 225
Gávea, Rio de Janeiro, RJ, Brasil, 22451-900
{brunosantana, simone, julio}@inf.puc-rio.br

Abstract

Ways to deal with differences among users and hardware platforms have been investigated by several subareas in Computer Science. Despite these research efforts, even today we lack a systematic approach to deal with variations that reflect differences in user goals, needs, preferences, and strategies to achieve goals. In this paper, we explore the variability of the user's domain language *to promote the requirements engineers' reflection on the need to deal with variations and what strategies are adequate to deal with them further in system design, as it will be reflected in the user interface language. In the proposed approach, we take into account concerns involved in different contexts of use to better understand the dynamic of user goals variability during system usage, before considering how to achieve the user goals on user interface language.*

1. Introduction

If we change our focus from a software system and its internal functionalities to a broader view of context of use, one may realize that: (a) a system is used by different users with different needs; (b) a system is used by users with changing needs; (c) users work in changing environments; and (d) users work in different system environments (i.e. platform and infrastructure) [1]. There are different strategies to deal with these differences and variations, both during system design and usage. Common strategies to deal with variations during system **design** are:

- regarding the range of goals that are supported by the system:
  a) to design an "all-in-one" system, which aims to support most sets of user goals, needs and preferences; for instance, Microsoft Word with its hundreds of features. This strategy is to develop a single system for "anybody" in "any situation". However, the user interface usually becomes "bloated", confusing and being in the way of *a single* user with a specific goal in a specific situation [2]; and
  b) to design a product family of systems, which support different sets of user needs and preferences; for instance, (suites or) complementary applications by Corel and Adobe to work with images [3].
- regarding the variations in strategies to achieve each goal supported by the system:
  c) to design a system that allows for a single strategy to achieve each supported user goal, i.e., to reduce the variations into a strategy deemed as more common and suitable for most;
  d) to design a system which offers flexible support to the users` needs and preferences (i.e. the user can do the same thing in different ways); for instance, in many applications the user can copy something using any one of three different strategies: by typing Ctrl+C, by clicking on a menu item or by clicking on a toolbar button. It gives to user the freedom to choose how to obtain the desired results in a specific situation [4].

Common design strategies to help users deal with variations during system **usage** are related to the amount of control users have over the variations:

a) to design an *adaptive* system, i.e., a system able to automatically adapt its user interface and functionalities to specific user and situation; for instance, the personalized recommendations of Amazon.com. An adaptive system can offer a specific user interface and selected functionalities targeted at a specific user and situation with little or no effort by the user. However, the automatic adaptations may challenge the users` comprehension of the process and consequences of change, and reduce the user's control of the system [5]; and

b) to design an *adaptable* system, i.e., a system which lets the user modify the user interface and, in some cases, the system functionalities. For instance, the user can choose if Firefox will open or not pop-up windows, and where new pages will be opened – in new tab or new window. This strategy keeps user in control of the user interface and system functionality changes, so the user is better equipped to comprehend the process and consequences of each change. On the other hand, the user needs to learn how to interact with the system to adapt it and may need to do substantial work to achieve an adaptation [6].

It is important to note that the aforementioned strategies, either during system design or usage, are not mutually exclusive. In most systems, they are conjugated and some of them complement others very well. We can find many applications nowadays which apply one or more of these strategies. However, the design and development processes typically deal with variations in an ad hoc manner, i.e., they lack a systematic approach to deal with variations.

The problem in dealing with differences and variations in user needs and preferences has been investigated by several subareas in Computer Science. In particular, we are interested in research in both Requirements Engineering (RE) and Human Computer-Interaction (HCI).

In RE, variability has been investigated by the identification of alternative ways (sequences of actions) to achieve user goals[1] [7-10]. Each way to achieve a user's goal is usually associated with one or more softgoals. A softgoal describes a non-functional requirement which helps or hurts each possible strategy of achieving a user's goal [7]. In general, in RE the chosen way to achieve a user's goal is decided during system design, supported or not by computer tools (or algorithms), according to the prioritized softgoals in that moment. In addition, it is still not clear what concerns should be considered in this decision process, beyond the concern on "how to achieve a goal".

Traditionally, research in HCI pays attention to variability of users, goals, tasks and context of use [11,12]. This interest by variability has motivated the development and improvement of techniques to deal with variability, following strategies applied during system design [2, 4] and usage [5, 6], as discussed above. Nevertheless, there is still a gap between identifying variability and deciding how to deal with it.

In this paper, we argue that it is necessary to take into account other concerns beyond "how to achieve a goal", to better understand the dynamics of variability during system usage. We, thus, explore the variability of these additional concerns on the goal-directed user requests in terms of the **user's domain language**.

During interaction, users should be able to express these goal-directed user requests (and possibly some variations as well) through the user interface language (UIL) [13,14]. The UIL is the language through which the user and the system interact with each other. It is represented by the static and interactive user interface elements (widgets), interaction patterns and strategies, help and any other information made available to the users about the user interface and its usage. The concept of UIL has a precise meaning within semiotic engineering [13] and should not be confused with interaction or interface modeling languages or notations, such as, UsiXML (www.usixml.org) and other user interface description languages.

In our language-based variability analysis approach, the language we refer to is the user's natural language. During the analysis, we do not aim to define the application's UIL, because this would be premature at this stage, nor to define an artificial language designed to analyze variability. Instead, we investigate the user's language when users talk about the domain to analyze possible variations in the language and its use that will need to be accommodated in the UIL, elaborated later in the design stages.

With information about anticipated variability at hand, we promote the requirements engineers' reflection to help them to decide which strategy should be used to deal with this variability during the UIL design. As a running example, we apply the proposed approach in the variability analysis of a media player system.

In the next section, we describe the motivations and causes for variability in systems design. The third section describes the seven activities of the proposed approach, from eliciting information relevant to the variability analysis to the analysis itself. In section 4 we briefly present some design concerns related to the variations identified previously, and section 5 concludes the paper.

[1] Using requirements engineering terminology, "user goals" refer to the stakeholders goals. In this paper we focus the attention on stakeholders that will interact with the system, as such the term user is utilized.

2. Which Concerns Are Involved In System Variability Motivated by Use?

Before understanding which concerns are involved in variability, we need to understand what goes on during the use of a system. As illustrated by Figure 1, the use of a system occurs when a user is engaged in an interaction process with the user interface to achieve some goal within a context of use [12,13].

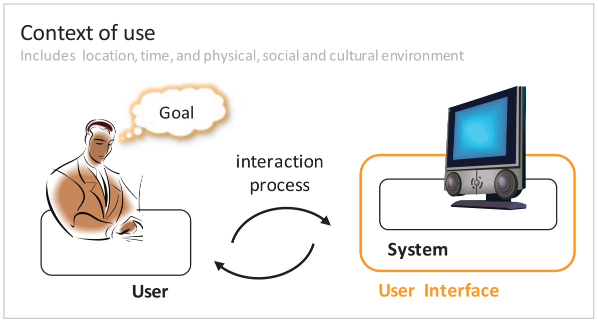
Context of use
Includes location, time, and physical, social and cultural environment
Goal
interaction process
User
System
User Interface

Figure 1. The process of use of system.

From the user's point of view, the system is the user interface, because the user interface is the only part of the system the user is in contact with (e.g., the user cannot access any system functionality if not via the user interface). The context of use includes the location, time, and physical, social and cultural environments where the interaction takes place. All these elements (and not only user goals) can influence the variability of system. Therefore, we need to investigate how these concerns (and their variations) are related with the variability of the user interface and the system functionalities. To elicit these concerns, we can use the following questions [15]:

Who participates in the interaction process? Examining the interaction process, we realize that both *user* and *system* (or the designer's deputy, in semiotic engineering terms [13]) participate in it. Regarding the users, we need to elicit information about their skills and preferences [8], as well as any constraints and special needs they may have. Regarding the system, we need to elicit information about the available hardware platforms (desktop, laptop, PDA, cell phone, etc.), input and output devices (mouse, keyboard, pen, touch screen, etc.), and infrastructure (network and internet access, disk space, etc.) in different contexts of use. At this stage, we are not making design or implementation decisions. We are just eliciting possibilities that will help anticipate variations. Only later should these aspects drive design decisions. For instance, it is not possible to download a file if there is no disk space available or if the network connection is down.

One should note that the system is not equivalent to users. It does not have its own intentions or free will. Instead, it behaves only as designed.

What are the participants' goals? The users' goals (i.e. the expected result of their interaction with the system) are traditionally investigated by requirements engineers [16, 7, 11]. The "system's" goals, on the other hand, are designed in later stages of development process to support the users' goals.

How can goals be achieved? The possible strategies to achieve user goals are also traditionally investigated by requirements engineers, and more recently their variability has also been taken into account [7-11, 16]. Later during design one or more selected strategies will be mapped onto action sequences on user interface.

When will goals be achieved? Besides common known time divisions, such as minutes, hours, days, months, and so on, the requirements engineer should investigate other relevant time divisions or intervals, such as seasonal intervals.

Where will goals be achieved? The interaction can occur in a set of places, such as home and work, or in a hierarchy of places, such as a room at a university, a campus, a city, a state, and so on. An environment analysis should investigate the physical (such as light and noisy levels), social (such as the possibility of learning to use the system with colleagues or when users are pressured to go fast) and cultural (as this culture works better with uncertainty than others) aspects of the environment that can interfere in interaction process [11, 12, 15].

The analysis should not only consider variations in one kind of element (for instance, among users or among environments), but also variations in time, with special attention to the frequency of change. For example, it is not sufficient to investigate that a specific user has such and such skills and preferences, because they can change in time, motivated, among other factors, by training or attitude. As the concerns involved in variability can change during system usage, it may not sufficient to deal with variability searching for "the best option" at design time for user X and context Y (privileging some softgoals over others) to design and develop the system according to this option. It is also important to consider strategies to deal with variability later, during system usage, as will be briefly described in section 4.

3. A Language-Based Approach to Explore Variability Concerns

To analyze variability, we need to go beyond "how to achieve goals?" concern. The concerns related to variability are typically described by softgoals [7-10], which qualify, positively or negatively, different ways to achieve user goals. In our variability approach, we decide to change our focus from a cognitive approach, where the interaction process is viewed as a sequence of user actions and interpretations [17], to a semiotic approach, where the interaction process is viewed as a signification and communication process [13]. Both the meaning of **what** the user requests directed by his goal to the system and **how** to express these requests at the user interface (i.e. the interactions) are important. However, the "what" should be defined before and then help determine the "how". This idea is reinforced when we remember that achieving a user goal results from a cooperative work between user and system which needs communication or exchange meanings and ideas. Our focus in this paper is on analysis of (the meanings of) what the user requests to the system, directed by his goals. Therefore, to understand how users elaborate his requests and what his intentions are, we should investigate how people exchange meanings and ideas.

The main entity or piece of information investigated in signification and communication processes by Semiotic is the sign. A *sign* is "anything that stands for (meaning) something to someone", such as words, pictures, and others [18]. The meaning of a sign is not static and evolves during time in a process called unlimited semiosis [19].

When we communicate with each other, we exchange signs (or ideas) according to some code or language. During the user-system communication (interaction), the user interface represents (or instantiates) a language used by both user and system to communicate with each other [13,14]. The user interface language makes systematic use of the signs encoded in the system. When the system is implemented, there is encoded in it *a* meaning of every sign which the system can "make sense" of. These encoded meanings command the system behavior. The meanings of all encoded signs in the system will always be partial and fixed, but because of unlimited semiosis, the meanings of all signs continue to evolve in the development team's minds after codification in (or implementation of) the system, as well as in the users' minds.

The user interface language must be as close as possible to the language used by users to talk about the domain [13, 14]. This will facilitate the production and interpretation of sentences during user-system communication (interaction). The concept of user interface language is similar to "application language" proposed by [20]. We prefer to use the term "user interface language" to remind us that this language will also be used by the users and not only by the application.

Therefore, the variability analysis in a semiotic approach is related to the variability in both the user's domain language and the user interface language. As mentioned in the introduction, the user interface language is defined later, mainly during interaction and user interface design. Given the amount of variability concerns, it is interesting to analyze smaller sets of variability concerns during several activities of the development process. This will reduce the amount of variability concerns analyzed at a time and will result in a user interface language designed for variability.

In the early stages of a development process, users and their goals are among the first identified elements. In line with [7-10], we propose to begin the variability analysis with the identified user goals. In our language-based approach, we propose a process to analyze the variability of user goals based on the user`s domain language (Table 1). The identified variability should be accommodated in the user interface language, elaborated later in the design stages. Although the activities in the process are presented sequentially, the proposed variability analysis process is actually incremental and iterative.

Table 1. A language-based variability analysis process.

Type of analysis	Activities
Lexical and semantic level	1. Elicit information of domain and contexts of use; 2. Identify and describe signs;
Syntactic and semantic level	3. Identify goal-directed user requests; 4. Rewrite goal-directed user requests as cases; 5. Review goal-directed user requests by means of systematic question asking;
Variability	6. Organize the signs in an ontology; 7. Analyze variability by means of systematic question asking.

3.1 Elicit information of domain and contexts of use

By means of interviews, questionnaires, observations and analyses of documentation and similar systems, the requirements engineer obtains information about the domain and the contexts of use, as discussed in section 2 [11,12].

3.2 Identify and describe signs

Given the previously elicited information, the requirements engineer identifies the signs that belong to the domain and that are candidates for being designed later into the application's user interface. In this variability analysis process, the requirements engineer should register certain information about each sign: name (and synonyms), description of the meaning that will be encoded in the system, restrictions to the possible values the sign may take on, a default value (if any), and the relations between the signs. These pieces of information can be registered in a structured representation, such as the Language Extended Lexicon [21], briefly presented here in a table. In a media player system domain, we can find some signs as the examples presented in Figure 2.

Play	
Meaning	to perform selected media files or media files from a playlist
Possible Values	-
Default Value	-
Relationships	Activated by the user. Has to be "previously buffered".
Synonyms	Go

Media File	
Meaning	Represents an audio or video file, with a title and duration time
Possible Values	Any media file which system can reproduce
Default Value	-
Relationships	Was recorded by an artist Can be part of an album Can be part of playlist and media library
Synonyms	song, video

Playlist	
Meaning	Represents an ordered list of media files chosen by the user
Possible Values	Any media file which the system can reproduce
Default Value	Empty list
Relationships	Can contain media files from media library Can contain media files from CDs, DVDs and MP3 Players
Synonyms	-

Figure 2. Partial descriptions of some signs in a media player system.

Registering this information for all signs which make up the user's domain language can seem excessive work at first. However, we should remember that the system will encode a meaning of all signs in the user interface language. As it is important that every stakeholder who produces or uses the system have access to (a description of) the encoded meaning of signs, representing this information in natural language facilitates human comprehension and helps establish shared knowledge about the application being designed.

What about words considered common sense? Should they still be described? Yes, they should. We should not forget that sign meanings, be they considered common sense or not, are all culturally-dependent, and many signs have multiple meanings. In addition, the encoded meanings of signs are arbitrary, resulting from the requirements engineers' and designers' semiosis processes that have arbitrarily stopped at some given moment. For example, the word "play" has more than one "common sense" meaning. In a media player system, the meaning of the "play" sign is in line with "to perform media files" and not with "to exercise oneself in diversion or recreation", nor with "to move freely within a space" (definitions from dictionary.com).

Moreover, the meaning of a sign keeps evolving in the stakeholders` minds, characterizing unlimited semiosis [19]. When we describe a sign meaning which will be encoded in the system, one stakeholder may compare the described meaning with the evolved meaning in his mind, as the development process proceeds. If the difference is significant, the stakeholders should (re)negotiate and review the meaning that will be encoded in the system.

Furthermore, the information about encoded signs will be very useful during subsequent activities in the development process, such as user interface design and help content development.

3.3 Identify goal-directed user requests

Taking into account the information collected about domain and context of use, the requirements engineer identifies the user goals that will be supported by the system [11, 16]. The requirements engineer can continue his work by analyzing different ways for users to achieve their goals, for example, by decomposing the user goals in AND/OR sugbgoal trees [7]. However, as we are interested in analyzing variability based on the user`s domain language to later support the design of the user interface language, it may be interesting to encode the user goal as a high-level request to the user interface, represented using the previously identified signs. This way, instead of representing the goal itself (e.g. Listen to song X), we represent the goal-directed user request, or simply user request, i.e., **what** the user wants the system to do to achieve his goal (e.g. Play X). We are not yet concerned yet with **how** to express his request to the system using the user interface language (e.g. selecting media file X and clicking on button ▶), which corresponds to design issues. In our media player system example, we can point out the following identified user requests:

- play media file;
- organize media files in a playlist: (a) create playlist; (b) add a media file to a playlist; and (c) remove a media file from a playlist; and
- skip the current media file.

In the next section, these user requests will be further characterized to illustrate our proposal for variability analysis.

3.4 Rewrite goal-directed user requests as cases

The next step in the proposed variability analysis is to rewrite the identified user requests to represent them in a way to facilitate the variability analysis based on user`s domain language. Like Liaskos and collegues [10], we propose to use Fillmore's cases [22] as a basis for characterizing user requests. The cases considered in this paper are:

Agentive (A) is a category of signs which represent agents who request an action identified by a verb. A sign of this category typically represents actors or groups of actors found in the domain, including the system(s)-to-be. In the case of the media player, for example, the sign "user" belongs to `Agentive` category in the sentence "a user requests the system to play a media file"[2].

Dative (D) is a category of signs which represent agents who will be affected by the action described by a verb. These signs typically represent actors or groups of actors found in the domain, including the system(s)-to-be. The sign "user" also belongs to `Dative` category in the sentence "system notifies user when the copy of media files from CD has finished".

Objective (O) is a category of signs which represent objects affected by actions identified by a verb. The signs "media file" and "playlist" belong to the `Objective` category in the sentence "a user plays a media file of a playlist".

Factitive (F) is a category of signs which represent object(s) or being(s) that result from the action or state identified the verb. In the sentence "a user creates a playlist", the sign "playlist" belongs to the `Factitive` category.

Instrumental (I) is a category of signs which represent instruments involved in the performance of an action identified by a verb. The sign "playlist" belongs to the `Instrumental` category in the sentence "a user burn a CD with a playlist".

Manner (M) is a category of signs which represent the manner (i.e. the mode, the intensity, the velocity, the force, and so on) by which the action identified by a verb is performed. The sign "volume" belongs to the `Manner` category because "a user plays a media file in volume 70%".

Location (L) is a category of signs which represent (virtual or real) spatial locations where the action identified by a verb is supposed to take place. The sign "playlist" belongs to `Location` category in the sentence "a user removes a media file from a playlist".

Temporal (T) is a category of signs which represent the duration or frequency involved in the action identified by a verb. The abstract sign "play duration" belongs to the `Temporal` category as it is numerically represented in the sentence "play media files of a playlist for 30 minutes".

The general format for representing a user request is: `Verb[A(),D(),O(),F(),I(),M(),L(),T()]`. Since we do not have more than one user role, we will assume `A(user)` for all sentences. In the case of the aforementioned user requests, we have:

- play a specific media file: `play[O(media file)]`
- organize media files in a playlist;
  a) create playlist: `create[F(playlist)];`
  b) add a media file to a playlist: `add[O(media file), L-to(playlist)];`
  c) remove a media file from a playlist: `remove[ O(media file), L-from(playlist)];`
- skip the current media file: `skip[O(media file)].`

Each slot defined by a case reduces the search space of possible signs which can occupy the slot, and thus may facilitate the variability analysis, as we will further discuss in section 3.6.

3.5 Review goal-directed user requests by means of systematic question asking

When we describe a user request in a natural language sentence, it is common to summarize and omit some related pieces of information (or concerns). Sometimes, we do this consciously when we know (or remember) more than we describe. In other times, we do this unintentionally because we do not know (or remember) related pieces of information about the user request. These omitted pieces of information about a user

[2] In this system, there is a single user role, but in multiuser applications one would typically find different roles occupying the agentive and dative cases.

request can be important to help analyze the system variability.

The systematic question asking technique [23] has been explored to help remember and discover related pieces of information from summarized sentences. It "*allows cognitive scientists to examine the content and structure of the information that is linked to a particular concept in memory. (...) The answer to a question about a component in a narrative can reveal an otherwise hidden connection to an idea that is critical to understanding that component*." [p. 249].

For every identified user request, we propose to use the systematic question asking technique to help to remember or discover additional cases (or roles of signs as discussed in section 3.4) related with a user request sentence. If any other piece of information related to a user request is revealed, the requirements engineer should expand the user request both in natural language (section 3.3) and in cases format (section 3.4). If new signs are revealed, they should be identified and described (section 3.2) before proceeding with the variability analysis. Based on Fillmore's cases [22], we propose the following questions for systematic question asking to improve the requirements engineer`s comprehension about user requests and related signs:

Agentive: Are there other agent roles who cooperate with the performance of this action? Who are they?

Dative: Are there other agent roles affected by this goal? Who are they?

Objective: Are there other kinds of objects affected by this goal? What are they?

Factitive: Are there other kinds of objects resulting from this goal? What are they?

Instrumental: Are there other kinds of objects used as instruments to achieve this goal? What are they?

Manner: Are there other kinds of manners to achieve this goal (considering the same action)? What are they?

Location: Are there other kinds of (virtual or real) spatial locations to be taken into account during the achievement of this goal? What are they?

Temporal: Are there other kinds of duration or frequency involved in the achievement of this goal? What are they?

For each user request sentence, the requirements engineer should answer all these questions. Again, because we are dealing with a single user system, all Agentive cases will be considered to be a non-varying user.

We describe below the application of the systematic question asking technique on the user requests of the example media player system:

- play a media file – `play[O(media file)]:`

`Dative`? People close to the system can listen and watch the media being played, but the system is not aware and cannot handle this.

`Objective`? A media file can belong to one or more playlists, and thus the case `O(playlist)` may be added.

`Factitive`? This action produces change the state of current media file, from paused or stopped to playing. It also changes the current media file position. So, we may add two more cases: `F(state=playing)` and `F(media file position)`.

`Instrumental`? Just `media file`.

`Manner`? A media file can be played with different volume intensity, in a specific speed, and with a specific equalization configuration. Then, it is necessary to add three more cases: `M(volume)`, `M(speed)` and `M(equalization)`.

`Location`? `Temporal`? None.

The expanded sentence of this user request is: user plays a media file of a playlist, with a certain volume, speed and equalization settings according to the current position of the media file, changing state of media file – `Play [O(media file), O(playlist), M(volume), M(speed), M(equalization), O(current position of media file), F(state of media file=playing)]`.

- User removes a media file from a playlist – `Remove [O(media file), L(playlist)]:`

`Dative`? None.

`Objective`? Just `media file`.

`Factitive`? `Instrumental`? `Manner`? None.

`Location`? Just `playlist`.

`Temporal`? None.

This user request does not need to be reviewed.

- User skips the current media file `- skip[O(media file)]:`

`Dative`*?* None.

`Objective`*?* Beyond the current media file, this goal affects another media file which will be played in the place of the current media file. To represent this we will replace `O(media file)` with `O(media_file_A)` and `O(media_file_B)`.

`Factitive`? The media file in question is playing and will be substituted by another media file which will be playing too. So, we can add the cases: `F(stopped-A), F(playing-B)`.

`Instrumental`? The next media file depends on the order sign (for example, sequentially or shuffle) and the continuity sign (indicates if the system should continue playing the playlist from first media file when the last media file has finished) on a playlist. Then, it is necessary to add three more cases: `I(playlist)`, `I(order)` and `I(continuity)`.

`Manner`? `Location`? `Temporal`? None.

The revised description sentence of this user request is: user skips the media file being played to the next one in a playlist, according some order and continuity – ship `[O(media_file_A), F(stopped-A), O(media_file_B), F(playing-B), I(playlist), I(order), I(continuity)].`

Even in a widely known domain, such as media player systems, we tend to summarize the description of user requests. These examples showed that the systematic question asking activity may reveal important related pieces of information of a user request. The cases identified by the systematic question asking will be useful to analyze variability, as we will show in the following.

3.6 Organize the signs in a domain ontology

We propose to explore sign substitutions to analyze variability. How can we find a sign which can be put in the place of another? A good way to do this is to search for another related sign according to some criteria. Thus, we propose to organize the signs in an ontology [24].

Given an ontology, signs that are candidates for substituting other signs may be found by following a few simple traversal movements:

- siblings: when a sign A and a sign B have an is_a relationship with sign X, than A is a candidate for substituting B in a variation, and vice-versa (e.g. a audio file may be substituted by a video file);
- specialization/generalization: when a sign A is_a sign B, then A is a candidate for substituting B in a variation, and vice-versa (e.g. a audio file may be substituted by a media file)
- related via a synecdoche: when a sign A is_part_of a sign B, sign A is a candidate for substituting B in a variation (e.g. a media file may be substituted by a media collection); if the verb accepts multiple elements as an argument, B can also be substituted by the collection of signs that are part_of B (e.g. a media collection may be substituted by the set of media files which comprise it);
- metonymically related: when a sign A is related to sign B through a metonymic relation [26] (e.g. located in; produced by), it is a candidate for substituting B (e.g. media files can be substituted by the MP3 player that contains them).

Generic traversals of the ontology, however, may bring about undesired results (e.g. substituting media file for user). Besides constraining the traversal by assigning weights or penalties to each kind of relation, it would be useful to further characterize the roles each sign can play in the sentences that describe potential user goals in the domain. We propose to have all signs related (directly or indirectly, via is_a relations) to a case. For instance, media file "can play the role (case)" `Objective` but not `Location` (therefore, there might be a can_play_role relation linking media file and `Objective`). This way, one would be able to copy media file to a certain location, but we cannot consider the media file to be a location per se, constraining the set of variations to be generated. The cases that define the roles each sign may play are identified in the user goals' definition.

An ontology which organizes signs can be reused in other systems in the same domain, with the necessary care to review the meanings of signs, of course. Even in the same system, as in the case of the media player being investigated, the pieces of information about signs can be reused. For example, pieces of information and relations of "media file" and "playlist" signs are used by both "play" and "remove" user goals. In an approach of user goals decomposition in AND/OR subgoal trees this is not possible, and it is necessary to have similar trees to represent them.

3.7 Analyze variability by means of systematic question asking

Having identified the signs, organized them in an ontology, and explored cases sentences that express user requests, we should conclude the variability analysis of user goals by exploring possible variations in these sentences in different contexts of use.

From the initial motivation for variability cited in the introduction [1], we can identify four dimensions to analyze variability: (a) variability from user to user; (b) variability of a single user in time; (c) variability from one context to another; and (d) variability from one platform to another ([15] proposes similar dimensions). We propose to analyze the variability of each user goal sentence as a function of these four dimensions – time, user, context and hardware platform – by means of systematic question asking [23]. Thus, using information obtained in elicitation discussed in section 2, the requirements engineer should answer these questions for each user request:

What can vary? Why? A user request written as cases is very useful to help us investigate this question. Given a case sentence of a goal-directed user request, what can vary is: (1) the meaning of the verb, (2) the set of cases associated with the verb, (3) different signs (sign-types according to Eco [19]) for substituting each case in the request (the natural candidates are signs resulting from transversal movements in the signs ontology), and (4) different values (sign-token values according to Eco [19]) which a sign can assume in the sentence (the natural candidates are the identified possible values of the sign in section 3.2).

When can it vary? In which situations can it vary? According to what does it vary? With what frequency? Why? Here the requirements engineer should consider the four mentioned variability dimensions [1,15]:

- Can it vary **in time** for the same user? (Related with "Who" question to user in section 2.)
- Can it vary **from user to user**? (Related with "Who" question to user in section 2.)
- Can it vary **from one context to another**? (Related with "Where" and "When" questions in section 2.)
- Can it vary **from one hardware platform and infrastructure to another**? (Related with "Who" question to system in section 2.)

How much can it vary? It is possible to enumerate variations? What are they?

Is it possible to omit some cases? What would be the default value? Could the default value be different or vary? When and why?

What could be the impact of the variation on the use if the system does not deal with it?

How many people could realize this variation? One user, few, many? Why?

These questions are based on interaction concerns presented in Figure 1 and on research on adaptive user interfaces presented in [1].

The natural and expected way to record the answers to and the thoughts fostered by these questions is using natural language description. However, if the requirements engineer wants to, he can organize some concerns of these questions in a graphical way using trees with AND/OR hierarchies to facilitate his reflection, as presented in Figure 3. (The dashed rectangle in Figure 3 represents an optional case.)

In spite of the similarity with the AND/OR decomposition of goals in subgoals [7-10], this representation has significant differences. First, every level of this hierarchy is associated with some type of concern (verb, case, sign-types – or simply signs, and sign-token values – or simply sign values) used in our language-based approach to variability analysis. Second, the hierarchy between the verb and case levels represents a relation established by user request; while the hierarchies between case–sign-type and sign-type–sign-token value are relations that reduce the scope of possible signs in direction of its instantiation. These meanings of the hierarchical levels are very different from decomposition of a user goal into subgoals.

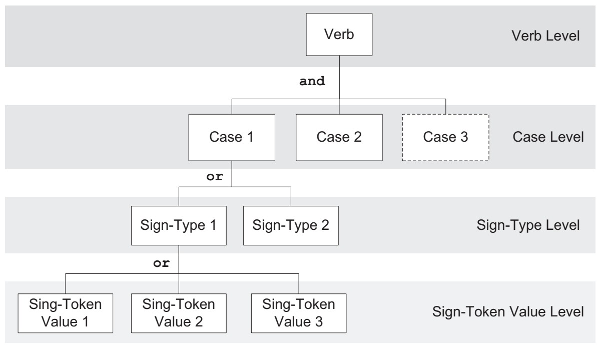

Figure 3. A representation for language-based variability concerns of user goal sentence.

We present below the application of the systematic question asking technique to analyze the variability of two user requests in the example media player system.

Figure 4 shows part of the variability concerns for the "**What can vary?**" question about the user request "user plays a media file of a playlist, with a certain volume, speed and equalization settings according to the current position of the media file, changing state of media file". In particular, we explore the cases `O(media file)`, `M(volume)` and `M(equalization)`.

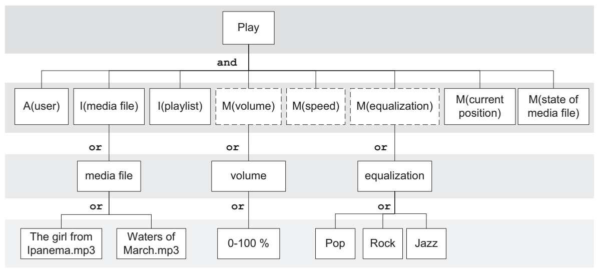

Figure 4. Part of the variability concerns of "Play".

Regarding "**When can it vary?**" concerns, we realize that:

Variability of a single user in time: a user can choose media files according to his preferences at moment. For example, he may want to listen to a specific music. Or, when he is happy, he could prefer to listen to a set of dance songs. Thus, we may explore and offer to the users different ways to choose media files at different moments and decide how the user alternates between them. The sign ontology can indicate some criteria to choose media files. In particular, transversal movements via a synecdoche ("part_of") and metonymical relations can point out good criteria, such as media files from an album or interpreted by an artist, respectively.

Variability from one context to another: Anna usually listens to music loud at home using her notebook, but at her university campus, she prefers to listen to music less loudly so as not to disturb her colleagues. Therefore, not only is it necessary to provide users with the volume control, but it may be interesting to create usage profiles for easy configuration of the media player in different contexts. In addition, some variations in context occur abruptly, such as a phone ringing, so a way to turn the volume down completely (mute) may be considered.

Variability from one platform to another: When Bob used his old and very simple speakers, he was not concerned with the equalization of songs being played; but when he bought new "super speakers", he discovered that adjusting the equalization can make a lot of difference, and he liked the results. He also noted that different songs required different equalization settings. Then, it would be interesting to save the equalization settings for a song or set of songs.

About of "**How much can it vary?**" concerns, we realize that: (1) The possible media files are potentially any file which a user can access from CDs, DVDs, pen drives, hard disks and so on. The possibilities are so many, that it is impractical to enumerate them. Also, it is very common for a user to change his mind and choose to listen to different media files from his initial selection. Therefore, it is better to keep the user in control of the system and allow him to choose what media files should play. (2) The volume can vary from 0 to 100 percent and can do so very frequently. (3) There are some predefined equalization settings, such as, pop, rock, dance, and so on; and, if user wants, he can change manually the equalization settings frequency by frequency. We do not expect that user change manually the equalization settings so frequently.

About "**It is possible to omit some cases?**" concerns, we realize that there is no default value to media file, thus, the user must choose media files to play them, at least indirectly putting a CD (media files from CD) to play. On other hand, it is not necessary to impose to user the necessity to inform the volume or equalization settings; because the system can presuppose default values (e.g. 100% for volume and 50% for each frequency of equalization settings). When the user adjusts the volume or equalization settings, two variations may be considered: a) it is a one time adjustment, and the next time the user runs the system the default values are restored, or b) the user is adjusting the system's default values, i.e., its presuppositions to adequately "interpret" the user request "play media files", without mentioning volume nor equalization settings. In the media player system, the second option is usually preferred. However, it raises another issue: the user need to be able to restore the initial default values, in case he inadvertently changed an important setting and does not remember what the initial configuration was.

Regarding "**What could be the impact?**" concerns, we realize that the user experience [12] can be seriously injured by playing inadequate media files with inadequate volume, and can cause embarrassment in a social environment. Inadequate equalization settings can also do that, but typically with less impact.

The "**How many people could realize this variation?**" concerns do not apply directly to our example, because it is a mono-user application. However, if we think of different user profiles, this question can uncover interesting issues.

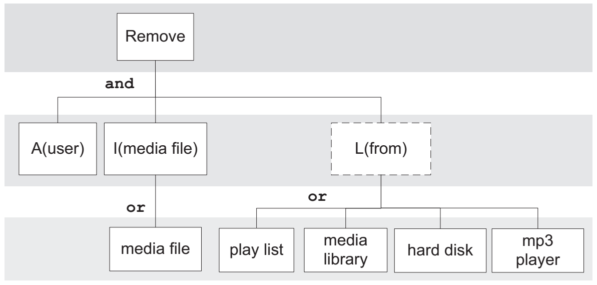

Figure 5. Part of variability analysis concerns of "Remove".

Figure 5 shows partial variability concerns for the "**What can vary?**" question to the user`s goal "remove a media file from a playlist". In particular, we explore the case `L(playlist)`. By means of transversal movements in the signs ontology, especially by siblings and specialization/generalization relations, we found three signs which can be placed in the slot of playlist: media library, hard disk and mp3 player. It is important to note that if the user puts a playlist or a hard disk sign in the slot of case `L(from)`, the meaning of the verb `remove` may be significantly different. It may be the case that a) the user wants to *remove the references* to media files from the playlist, or b) the user wants to *delete the media files* from the disk.

Regarding "**How much can it vary?**" concerns, we realize that: The possible media files are potentially any file which a user can access from hard disks and pen drives, which makes it impractical to enumerate them. Also, it is very common for a user to change his mind and decide to remove different media files at different moments. The same is true for `from` possibilities, both regarding enumeration and user`s decision.

Regarding "**Is it possible to omit some cases?**" concerns, like with the "play" user request, the user must choose media files to remove them. If the `from` sign is omitted, several interpretations may hold when requesting "remove file A": a) remove the file from the current playlist; b) remove the file from the media library, but keep it in the physical storage; c) delete the file. Even when the requirements engineer and designers believe there is a preferential interpretation (such as remove from current playlist), it is important, because of the possible variations, to inform users of the precise encoded meaning, because a misunderstanding can have a serious impact, such as the undesired loss of a media file.

4. How To Deal With The Identified Variability?

In the previous activity, the requirements engineer investigated answers to questions about anticipated variability of goal-oriented user request and reflected about its foreseen impacts during use. Once acquired this information, how to deal with the identified variability? In this paper we propose some questions to help the requirements engineer decide if it is necessary to deal with the identified variability and what strategies should be used. These decisions about variability should guide the user-system interaction and user interface design in further activities of development process.

Given a user goal, for each possible identified variation, it is important to decide:

Is it necessary to deal with this variation in the system? Even if the requirements engineer decides not to deal directly with the variation, the user interface designer will need not only to acknowledge the possibility of variation, but also to deal with users' possible expectations regarding the variation.

If the requirements engineer chooses deal with the variation in question, he should decide "**What strategy should be used to deal with it?**" taking into account the strategies discussed in introduction of paper. Thus, this question raises other questions:

- Will there be a single product supporting all variations for all users? Or a product family, where each product supports a subset of goals?
- Will the user be able to select a variation and keep it as the default?
- Will the user be able to create configurations (profiles) of settings to apply in different contexts?

  Will the system take care of adapting the system for the user, as deemed appropriate according to some predefined set of models and rules?

5. Concluding Remarks

We presented a language-based approach to variability analysis of goal-directed user requests in terms of user's domain language. We take into account concerns involved in different contexts of use, before considering the possible sequences of actions on user interface to make these requests to the system. In the later activities of development process, a user interface language [13,14] will be designed to allow users to express their requests and accommodate the identified and chosen variability.

The basis of our language-based approach is the analysis of signs [18] which will further compose user interface language. Most of the signs in the user interface language should be somehow related to the user's vocabulary and domain.

Using Fillmore's cases [22] to help us analyze variability is not a new idea [10], but this work advances in that direction and proposes to use the cases as a constraint mechanism for traversing a sign ontology when exploring possible variations. This categorization can be reused in similar domains with the necessary care to review the sign meanings. When this work frames the variability problem in terms of the variability of signs in user request sentences, we are able to explicitly explore not just variations of sign-tokens as in [10], but also variations of sign-types, of the sign-types (or cases) associated with verb, of verb, and of their meanings. Furthermore, analyzing variability in terms of possible goal-directed user requests sentences facilitates the communications between stakeholders about it.

The decision of how to deal with identified variability is sometimes made by a computer tool according some privileged softgoals [7-11]. Our work promotes the requirements engineers' reflection to help them decide which strategies should be used to deal with the identified variability during the user interface design activity. It guides the requirements engineers' reflection by means of a set of questions about variability applied to user request sentences, according with the proposed dimensions of variability [1,15].

In a preliminary study using these questions, we could verify that (1) an expression of the user request during the interaction can omit some cases or signs (i.e. a user can take a few things for granted that another user would not); (2) it is important to think about the impact of an identified variation before design and implementation; and also (3) it is important to think about which users can experience an identified variability.

The decisions resulting from the proposed language-based variability analysis depend little or nothing on the possible action sequences to achieve a user goal, on a specific solution of user interface design and on a specific technology to implement the user interface. This occurs because they are made in terms of language for user-system communication (interaction), and not in terms of actions at the user interface.

As future work, we need to improve the support for the requirements engineers' reflection towards interaction and user interface design solutions to deal with the identified variations. It is also important to conduct experiments with the proposed approach to analyze variability with different systems and domains, with special attention to the difficulties and facilities which the requirements engineer may encounter.

6. Acknowledgments

All authors thank CNPq for the financial support to this work, and reviewers for valuable comments to improve our approach enunciation. The first author also thanks FAPERJ for the financial support to his Doctoral program.

7. References

[1] Kühme, T., Schneider-Hufschmidt, M., "Introduction". In Schneider-Hufschmidt, M., Kühme, T. and Malinowski, U. (eds.) *Adaptive User Interfaces: Principles and Practice*. Elsevier, North-Holland, pp.1-9, 1993.

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

[3] A Framework for Software Product Line Practice - Version5.0, http://www.sei.cmu.edu/productlines/index.html.

[4] Nielsen, J. "Heuristic Evaluation". In Nielsen, J. and Mack, R.L. (eds.), *Usability Inspection Methods*. John Wiley & Sons, New York, 1994.

[5] Kühme, T. and Malinowski, U. (eds.) *Adaptive User Interfaces: Principles and Practice*. North-Holland, Elsevier, 1993.

[6] Lieberman, H., Paternò, F. and Wulf, V. (eds.) *End User Development*. Springer, v. 9, 2006.

[7] Mylopoulos, J., Chung, L., Liao, S., Wang, H. and Yu, E. "Exploring alternatives during requirements analysis". *IEEE Software*, 18(1):92–96, 2001.

[8] Hui, B., Liaskos, S. and Mylopoulos, J. "Requirements Analysis for Customizable Software: a Goals-Skills-Preferences Framework". In Proceedings of the 11th *IEEE International Requirements Engineering Conference*, pp. 117-126, 2003.

[9] González-Baixauli, B., Laguna, M.A. and Leite, J.C.S.P. "Aplicación de un Enfoque Intencional al Análisis de Variabilidad". In Proceedings of the 8^{th} *Workshop on Requirements Engineering*, Porto, Portugal, pp. 100-111, 2005.

[10] Liaskos, S., Lapouchnian, A., Yu, Y., Yu, E. and Mylopoulos, J. "On Goal-based Variability Acquisition and Analysis". In Proceedings of the 14th IEEE International Conference on Requirements Engineering. pp. 76-85, 2006.

[11] Hackos, J. T. and Redish, J. C. *User and task analysis for interface design*. New York, NY, John Wiley & Sons, 1998.

[12] Preece, J.; Rogers, Y.; Sharp, E. *Interaction Design: Beyond Human-computer Interaction*. New York, NY: John Wiley & Sons. 2002.

[13] de Souza, C. S. *The Semiotic Engineering of Human-Computer Interaction*. Cambridge, Mass., MIT Press, 2005.

[14] de Souza, C.S., Barbosa, S.D.J. and Silva, S. R. P. Semiotic Engineering Principles for Evaluating End-User Programming Environments. Interacting With Computers. v.13, p.467 - 495, 2001.

[15] Sutcliffe, A., Fickasm S. and Sohlberg, M.M. "PC- RE: a method for personal and contextual requirements engineering with some experinence". *Requirements Engineering*, Vol. 11: 157-173, 2006.

[16] Dardenne, A., van Lamsweerde, A. and Fickas, S. "Goal-directed requirements acquisition". *Science of Computer Programming*, 20(1-2):3–50, 1993.

[17] Norman, D. "Cognitive Engineering". In Norman, D. and Draper, S. (eds.) *User Centered System Design*. Hillsdale, NJ: Lawrence Erlbaum, pp. 31-61, 1986.

[18] Peirce, C.S. (1931-55) *Collected Papers*. Cambridge, Ma. Harvard University Press. (excerpted in Buchler, Justus, ed., Philosophical Writings of Peirce, New York: Dover, 1955).

[19] Eco, U. *Theory of Semiotics*. University Press, Bloomington, 1979.

[20] Leite, J.C.S.P. *Application Languages: A Product od Requirements Analysis*. Departamento de Informática, PUC-Rio, January 1989.

[21] Leite, J.C.S.P. and Franco, A.P.M. "A Strategy for Conceptual Model Acquisition". In Proceedings of the *International Symposium on Requirements Engineering*, pp. 243-246, 1993.

[22] Fillmore, C. "The case for case". In *Universals in Linguist Theory*, ed. E. Bach and R.T. Harms. New York, Holt. 1968.

[23] Carroll, J.M., Mack, R.L., Robertson, S.P. and Rosson, M.B. "Binding objects to scenarios of use". *International Journal of Human-Computer Studies*, Volume 41, Issues 1-2, Pages 243-276, July 1994.

[24] Gruber, T.R. "A Translation Approach to Portable Ontology Specification". *Knowledge Acquisition* 5: 199-220, 1993.

[25] Eco, U. *Semiotics and the Philosophy of Language*. Indiana University Press. Bloomington IN. 1984.