Nimble

Jeroen Jillissen, Xiwen Cheng
LIACS*, Leiden University, Niels Bohrweg 1, Leiden, The Netherlands
jeroen@jillissen.com, xcheng@liacs.nl

Supervised by Y. Zhang, F. Verbeek
yanju@liacs.nl, fverbeek@liacs.nl

January 21, 2009

Abstract

The majority of published applications are not friendly enough for the vast majority of casual users. Guidelines applied to interface design are extremely saturated and rather complex for the novice users. This project is an attempt on creating a context-awareness interface to communicate with specialized applications based on ambiguity computing.

1 Introduction

In this paper we discuss our solution on how to help novice users to use the computer and utilize its applications without the need to really learn how to use these applications. By putting the focus on the task of the user instead of on the way an application should be operated, we were able to create a layer on top of existing applications which makes it possible for the user to formulate their own task to perform the desired task.

2 Problem definition

We often see novice computer users struggling while using the computer. These users often do not have the time, knowledge and/or interest to figure out how certain applications work or how to fulfill a task using the computer. This is a pity since a computer should empower users instead of being a barrier to get things done. This leads to the question:

"Can we make an interface which helps a (novice) user to complete a task in an intuitive and easy way?"

To tackle this question one could consider the following two general approaches.

*Leiden Institue of Advanced Computer Science

2.1 Focus on the application, improve existing applications

One approach is to improve existing applications in a way so that they might become easier and more intuitive to use by novice users. Although it might be a logical solution, unfortunately it is not a feasible solution since it is not possible to change every single application that is out there.

2.2 Focus on the task, looking at the task the user wants to perform

By looking at what the user wants to do instead of what an application allows the user to do, one could create a 'layer' on top of existing applications which could be used to use the functionalities of these applications in a way the user understands (simplification). This way the applications do not have to be altered in any way. A downside of this approach is the fact that every possible functionality should be 'connected' to this layer which is hard to do, unless it happens on an *Operating System* level.

3 Centralized cognitive interface

Instead of letting a user work with an application they do not know or do not know well, we haven chosen to take an approach where we focus on the task of the user, instead of the application they would use to perform the task. By letting the user express the task they want to perform, for example listening to a certain song or visit a certain website, we want to eliminate the need to understand the applications needed to perform the desired task.
To make the execution of a task as intuitive as possible the user should be guided as much as possible, in an unobtrusive way. The guidance which is offered to the user should always be optional.

Given the approach described above, we have made a prototype where we focus on the task of the user to see if we could make an intuitive system to complete the task of the user. By creating a command line interface which is able to interpret the command of the user, the user is able to formulate a command for a task they wish to accomplish, for example playing music using the command *Play music from The Stones* or *I want to listen to the song Yellow submarine*. Instead of figuring out how a music player works and where the music files are located on the computer, the user only has to bother with formulating the task they want to perform. The system will start the music player, locates the correct song or set of songs and starts playing. The initial screen can be seen in figure 1.

3.1 Command structure

A valid command is when valid object references or actions(depending on the action, one or more object specifications is required) can be derived. A formal structure definition is of this form:

$$I = [...][command_q][...][trigger_r][object_s][partial_t] \tag{1}$$

Where:
[] denotes optional
... can be a set of words
$command_q$ refer to recognizable commands
$trigger_r$ is used to help search for object references more efficiently

$object_s$ is a matching object reference
$partial_t$ partial object-name to be matched

Simple flexibility measurements are applied to extract valid data from the input string. Valid data are recognizable commands based on predefined command-action map and or objects located on the local system. Because of the non-strict command structuring conflicts might occur. The system will try to solve dependency between objects found and actions registered in a graceful manner. If the auxiliary attempt fails, a warning will be given.

3.2 Guidance

To help the user to formulate their tasks and make the system as intuitive as possible, the system provides the user with guidance. Guidance offered by the system is always optional and can help the user to quickly complete certain tasks by providing relevant information while the user is still typing their formulated command into the command line interface (figure 2). An example of this is a list of available songs from the Rolling Stones when the user is typing *I want to listen to the Rolli*.

3.3 Command variations

Since every user is able to formulate his or her own command the system should be flexible enough to interpret different variations in commands which essentially refer to the same task, like *I want to listen to the Rolling Stones*, *Play the rolling stones* or even *music the rollingstones*. This is done by extracting the 'object' (eg. the Rolling Stones) from the command which is used to determine the default action based on the type of the object if no action is explicitly defined. Simple typos are corrected automatically by the system based on soundex [1] similarity and pattern matching in MySQL [2]. At a later stage *Levenshtein-distance* [3] algorithm was applied to enhance error-tolerance. Figure 3 illustrates errorneous input being adjusted to the local environment (database).

3.4 Ambiguity

A (partial) command applied to the targets name (or title) can refer to different physical objects on the system (figure 4). This results in ambiguity which the user has to solve by either specifying the action or by selecting the appropriate object from the list presented by *Guidance*. If the user fails to do so, the system should give an error-feedback because of conflict in actions.

4 Results and Evaluation

Because Nimble tries to understand what the user wants by relating the user's desire to what is present on the system. The testgroup seems to be able to formulate their commands with their own words to interact with Nimble. Most of the inputs were processed correctly by our pseudo syntax module. The idea of assisting the novice user, in particular those that cannot blind-type, with Guidance was less successful as it contradicts their interaction fashion: All attention is focussed on the keyboard while typing. Those whom noticed Guidance's presence did use it to complete the task they were assigned to. However some attempts failed as Guidance did not support these

actions.

4.1 Feedback

The final prototype is the result of extensive user/task analysis, evaluation conclusions and feedback from our assistant and presentation. In the very beginning we ought to focus on multiple domains to illustrate the effectiveness of such concept. Based on the feedback we received from the presentation audience and assistant's approval we diminished the test-domain to playing music and some simple operations related to it. In the final development cycle more domains (photo and open url) has been added to give an impression the implementation is scalable. These domains are based on the knowledge gathered from the music domain evaluation, but have not yet been tested with actual users.
As our projectplan pointed out Nimble should accept Natural language as input; that gives the impression it is more effective (required effort during implementation) to use such a library to parse the input. Being able to parse human language is not enough. The main struggle is the interpretation of such input, as a human desire can be formulated in literally thousands of ways. Our focus is on the concept and how users will interact with such a system. Therefore we decided to build our own syntax module that checks syntax using a predefined set of recognizable definitions and alias-transformations in a database (MySQL). In the current phase of development the idea behind Natural language is strongly simplified in a very limited set of phrase compositions (dynamic processing of a language as complex as English, Dutch is even more complex, is a research on its own). The user-evaluations proved to be very useful in adjusting the system to the user's visions. For instance some users expect Nimble to play an album when the album cover is clicked. And another very "natural" behavior is to use (short) aliases for references. e.g. "stones" instead of "the rolling stones" or those less aware of the real spelling think it is "rollingstones". Based on these observations (and others) we were able to improve the program.

5 Conclusion and Discussion

Although Nimble does not support real natural language it is safe to conclude users do like such concept to communicate with a computer. Instead of having the user adapt to a computer, the roles can be swapped. Thus, composing a command equals writing down ones' desires in text-format; eliminating the learning curve imposed by current operating environments. The reason why such a system can be powerful is the fact the user does not have to know where the objects are located. They are only interested in the object being sought, the location does not matter. In conventional desktop paradigm it is necessary to organize the desktop regularly or whenever new items are placed on it. With the concept of Nimble files can be access while the user is unaware of the exact location of such a file, given they have the correct meta-data embedded.

Of course the question arises to what extend a human language interpreter should be able to understand what is going on. Imagine such an interpreter is possible, in some way or another it will have to map the interpretation back to an action.

Nimble acts as an extra layer on top of multiple applications *translating* inputs from the user to existing specialized programs. This implies the set of supported tasks is limited to the resources present on the computing environment. Also note the simplified tasks need to be mapped (or linked) correctly (somehow) to the application in question. There are boundaries for the supported tasks as commands for certain tasks can become rather complex in composition and therefore more error prone.

Finally we can conclude that the concept works, but at this moment it does not support virtually any imaginable input flawlessly. The implementation still requires a lot of work to actually call it a truly functional tool for mainstream usage.

6 Future Work

6.1 More user research

Although users seem to get how *Nimble* works, it might be very interesting to see how users react to the implementation of multiple domains. The last usertest only contained one domain (music) and therefore we do not know how the users will react on working with multiple domains.

6.2 Environment context

Given the knowledge gained from the current process and prototype, it might be interesting to see how one could improve on the context awareness of the system. Being able to automatically determine the context for an applicable command results in less specification needed from the user.

6.3 Make mapping more easy

In the current prototype every action has to manually be mapped to a functionality of a certain application. It would be intereseting to see if it is possible to come up with a system or protocol that makes it possible to automatically index and map functionalities to certain actions.

Appendix

Figure 1: The initial screen displaying a short description of Nimble

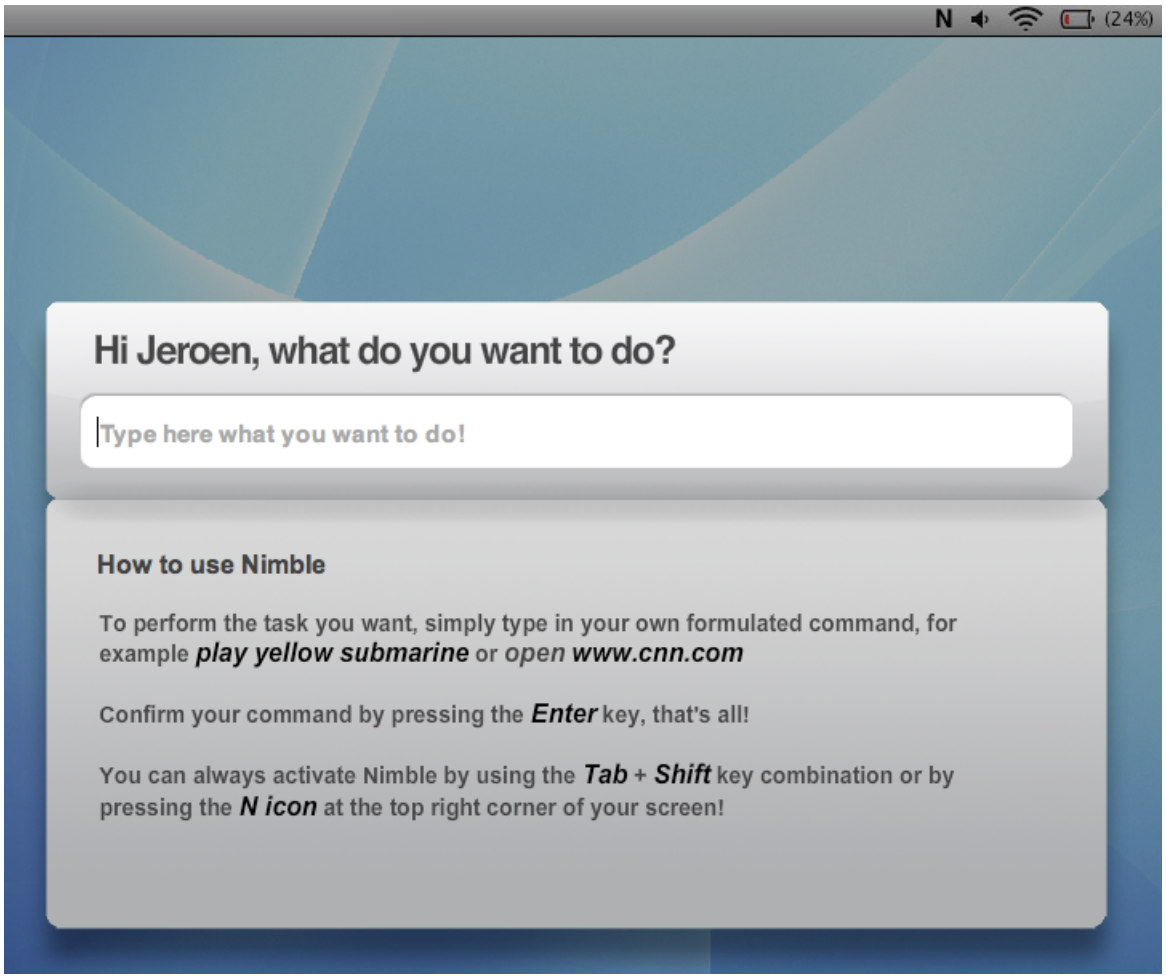


Figure 2: A default action is suggested by Guidance

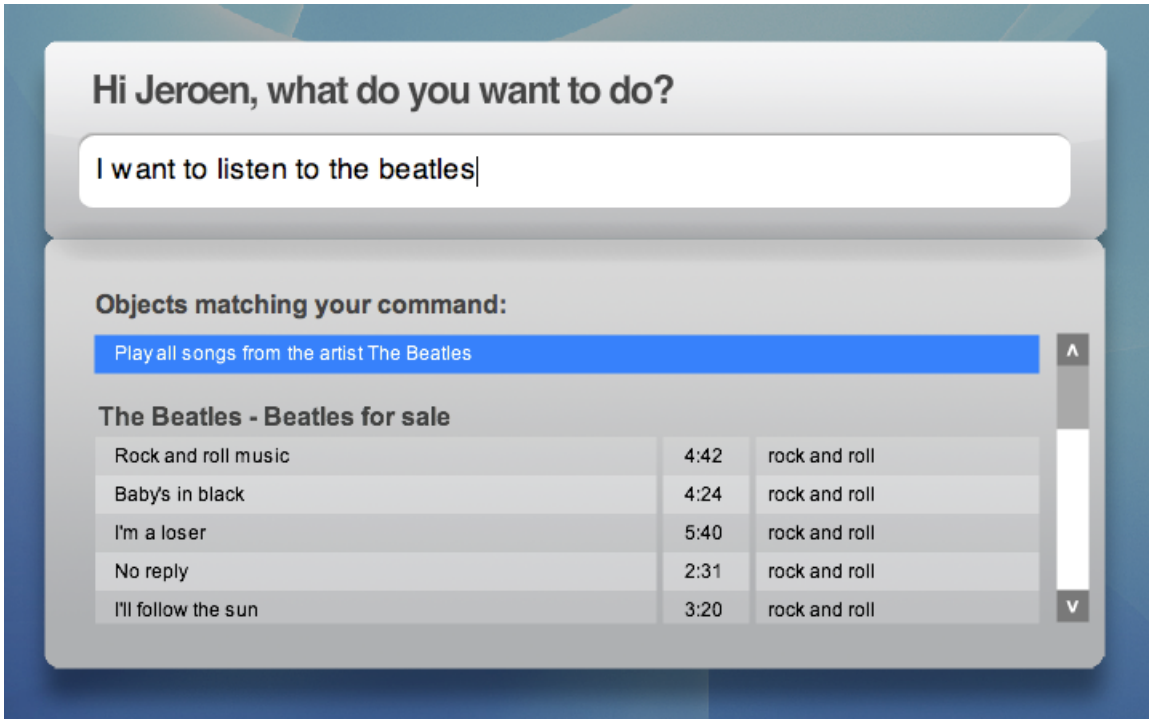


Figure 3: The system tolerates errors

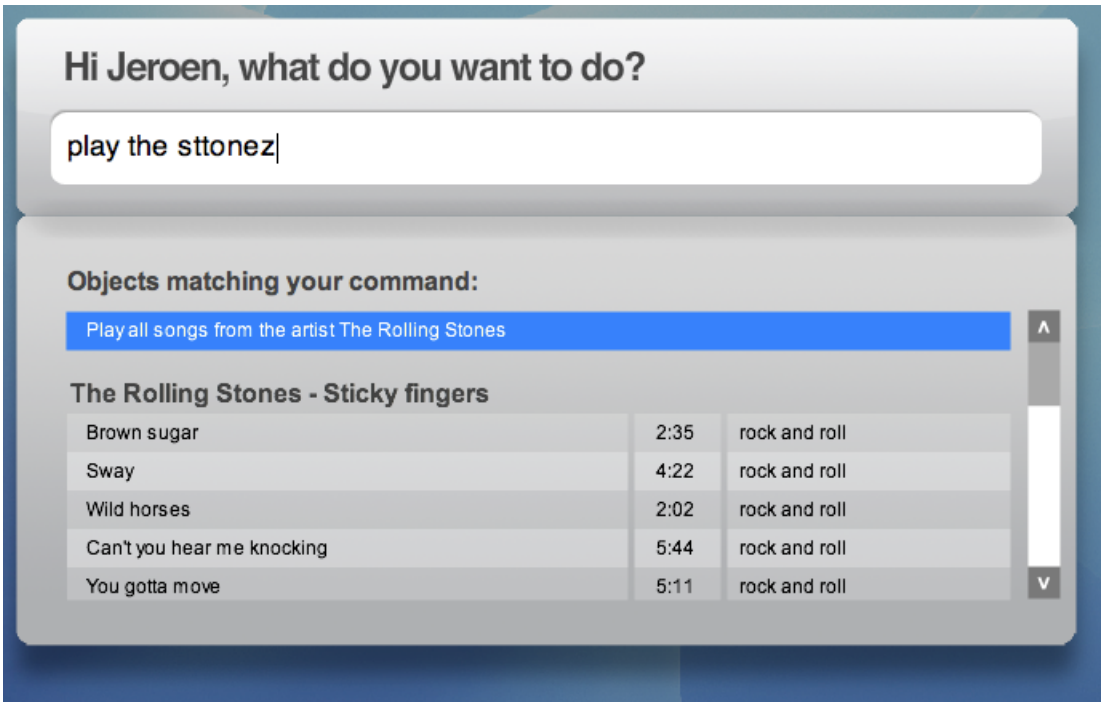
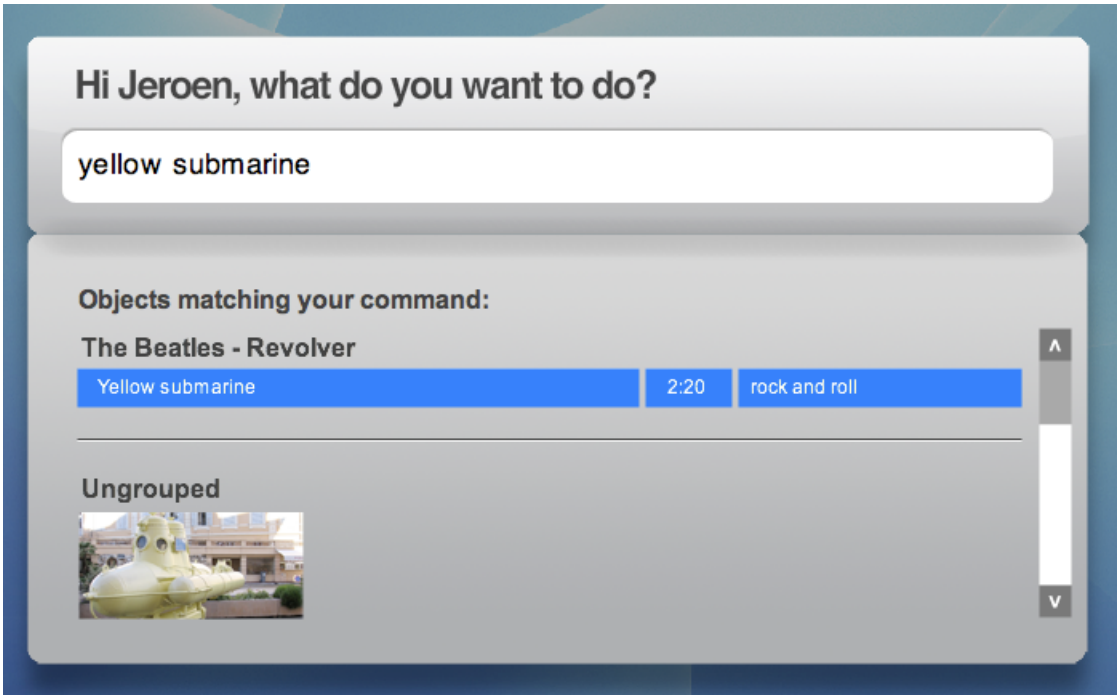


Figure 4: Multiple object-types are shown when object-name is not unique



References

[1] Soundex.
http://en.wikipedia.org/wiki/Soundex

[2] String Functions in MySQL.
http://dev.mysql.com/doc/refman/5.1/en/string-functions.html

[3] Levenshtein-Distance.
http://en.wikipedia.org/wiki/Levenshtein_distance