A Novel User Interface for Knowledge Base Browsing

Radu Adrian CIORA
Lucian Blaga University of Sibiu, Sibiu, Romania
radu.ciora@ulbsibiu.ro

Daniela GIFU
Alexandru Ioan Cuza University of Iași
Romanian Academy, Iași branch
Cognos Business Consulting S.R.L., Bucharest, Romania
daniela.gifu@info.uaic.ro

Carmen Mihaela SIMION
Lucian Blaga University of Sibiu, Sibiu, Romania
carmen.simion@ulbsibiu.ro

ABSTRACT

Intuitive user interfaces have been of great concern for GUI developers. The current research, who deals with their designing, faces the term intuitive constantly. The main question is how can the Interface be intuitive? For the moment, the researchers try to provide a very intuitive generic user interface that can be used in a variety of applications. In this paper we provide a solution that can model any applied ontology into a honeycomb menu. The hexagonal shape of the honeycomb has attracted the attention of humans for centuries. As a relevant consequence, the final user can browse any knowledge base very easily with the aid of this interface. Another useful feature is that programmers can take full advantage of semantic web technologies which can tailor results based on any knowledge base that is feed as input, without any need for code change, thus leading towards a panacea system.

Keywords: HCI, knowledge-based system, GUI, semantic web.

INTRODUCTION

Because the honeycomb structure is considered the strongest, yet, the lightest in various domains we considered it very useful in the ontology presented here. Moreover, it has inspired a lot of human innovations in architecture, transportation, mechanical engineering, chemical engineering, nanofabrication and recently, biomedicine (Zhang et al., 2015). Lyon and Colyvan (2008) discuss the role that the honeycomb proof in geometry plays in explaining the hexagonal structure of hive-bee honeycomb.

The work of Steve, Gangemi and Pisanelli (1997) introduce the metaphoric concept of ONIONS, in which the “onion leaves” symbolize interpreters in a local definition of an expression, linked to various paradigms. Thus, the challenge that remains is to develop a common ground for all the information that a knowledge-base holds, in order to map it into a discoverable source of information.
PROBLEM DEFINITION

The problem of optimum user interface design can be solved by considering the public that the interface addresses. It cannot be developed a universal interface that would suite both the younger and the older audience at the same time.

The work of Almendros-Jimenez and Iribarne (2005) suggests that use case model can help a GUI designer identify the needs for a user interface for a specific application. Solutions design were proposed by Samp and Decker (2010), as an optimization for decreasing the time needed for selection in displayed menus. Their paper suggests that radial layout is more intuitive for humans than the linear layout. Thus, the solutions that currently exist do not provide a panacea for GUI design.

SOLUTION PROPOSED

The honeycomb menu that we propose enables the final user to browse any knowledgebase very easily with the aid of its optimum interface design. Moreover, programmers also can benefit from this approach as the need for further expansions is not done by writing additional lines of code, but rather, by extending the underlying ontology. Thus, can be taken advantage of semantic web technologies, that interrogated correctly, can tailor results based on any knowledgebase that is feed as input, without any need for code change.

Figure 1: A Generic Honeycomb Interactive Menu.

As figure 1 shows the menu unfolds from the center root hexagon, towards its outer border. It is hierarchically structured in concentrically laid layers, starting from the central root and going step by step towards the outer layer/level of the knowledge-based system mapped.
Figure 2: A Small Sample of the Ontology Graph (Ciora 2009).

This solution can be considered a panacea per se as it is data independent, meaning that it can be applied on any data sets that it is tested upon.

In order to provide an example of how the ontology graph that sits behind a honeycomb menu would look like, we generated a small graph in Protégé. This is illustrated in Figure 2. Here, classes are represented by yellow boxes, whereas the actual instances are represented by grey boxes. The graph contains two sample applications and two possible inputs from the user and their corresponding interpretation – in the non-rounded squares as one can see in the ontology diagram. The applications are Microsoft Outlook and Eclipse and the inputs are the actual scan codes generated by the user input for mouse scroll and key down events: 522 and 256 scan codes respectively.

In order to prove our concept, we used an existing ontology called Context, developed by Ciora (2009). It was used for assessing students using the newly introduced honeycomb menu. It models the human computer interaction. The taxonomy of the most basic categories of particulars in our ontology is depicted in Figure 3. The categories are supposed to be mutually disjoint and to cover the whole domain of particulars (Monaghan et al 2019). The ontology has three main branches (Figure 3):

- The application branch – which creates a hierarchy of applications based on the application type (ex. integrated development environment (IDE) – Eclipse, e-mail client – Microsoft Outlook);
- The interface branch – which maps the input events raised by the input devices (mouse & keyboard); for example, we map the 522 and 256 interrupts which correspond to mouse scroll and key down events in Windows;
The activity branch – is the main branch in the ontology, it maps the activities of a user based on both the active application and interface currently being used for a certain task. For instance, if the Microsoft Outlook application is active and the 522-mouse scroll event occurs then we can create associated triples to add to our graph to log the application - Microsoft Outlook and the event - mouse scroll. The resulting graph provides a picture of the user’s activities over time and allows interesting queries to be performed on it using SPARQL to provide views from numerous angles. For our special case the resulting graph contains just one node with the value: “User reads an e-mail in Microsoft Outlook”.

The solution proposed was implemented in Python and tested on a Windows system. We took advantage of the operating system’s APIs: the OLE automation interface, that is based on the Component Object Model (COM).

When people discuss COM objects, they are often talking about only one side of COM – using automation objects. Automation objects are objects that expose a programmable interface that can be used by another programme or environment.
The OLE automation interface’s primary goal is automating applications. We don’t focus on this part of the interface, but rather on the information that it can provide about the applications it connects us to. This information goes from the Word document’s author to the subject or the content of an e-mail.

A second interface we used was the hooking interface. A hook is a point in the system message handling mechanism where an application can install a subroutine to monitor message traffic in the system and process certain types of messages. The system supports many types of hooks, but our interest was limited in hooking the mouse and the keyboard. This interface gave us access to the raw input of the input devices; this was useful at some extent - meaning that we could find out what, where and when the user clicked something, but it would not give us the information which UI element was activated, is any, by a mouse or keyboard event.

In order to able to tell which UI element was activated we needed the aid of a third API called Active Accessibility. The Active Accessibility is meant to help accessibility aids, here our application, interact with UI elements of other applications and the operating system. Therefore, this interface allowed us to retrieve which UI element a user activated at a given time.

With the aid of these three interfaces we can capture all the actions a learner performs on a computer. We extended the tool with a logging functionality, so we could keep a history of the student’s actions. On top of this low-level tracking tool we started building an assessment tool, which has as input an XML document. The XML document contains the description of a task that a student must solve. A sample of such an XML document can be seen in Figure 4.

Our program sits in background and monitors the student’s actions. When the student closes the application, we present him the outcome. If he or she executes the tasks correctly – meaning if he completed all the necessary steps in the required order, the system tells the learner that he successfully completed the test. If he or she failed to do so, the system provides him or her with feedback on the errors he or she committed. It shows him or her, the steps where he or she has failed and provides an explanation why he or she has failed and shows him or her, the correct solution.

**RESULTS AND CONCLUSIONS**

The testing of the system was made on a live environment, where several applications were launched, like: Skype, Word, Outlook. We evaluated the system using several use cases from several points of view. The first angle was the accuracy of the system and the second angle was the exception testing. The evaluation from the accuracy point of view refers to how well our system
can assess a student on a task. We verified if it gives a failure result to the student even if the student successfully performs a task. We recorded and performed several assessment tasks, some of them described in the previous section, to see whether the system assesses us correctly. The system gave us positive feedback and it did not report any incorrect results. The accuracy is given by its simplicity, as it is anchored deep into the operating system’s APIs. The exception testing verified the stability of the application. With exception testing, all the error messages and exception handling processes are identified, including the conditions that trigger them. A test case was written for each error condition. We aggregated these tests in an exception test suite. Finally, we looked at the optimality/intuitiveness of the interface, where were compared the speed of completing a task using the honeycomb interface versus a linear interface. The conclusion was that the user completed the given tasks approximately 30% faster using the honeycomb interface then the linear one. From the results the main conclusion is that the honeycomb optimality does prove to be the most suitable for user interfaces.

REFERENCES

Almendros-Jimenez, J.M. and Iribarne, L. (2005). Designing GUI components from UML use cases. 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS’05), (pp.210-217).

Ciora, R.A. (2009). Improvements in Automated User Assessment for E-Learning Environments, Masters Dissertation, National University of Ireland, Galway.

Lyon, A., & Colyvan, M. (2008). The explanatory power of phase spaces. Philosophia Mathematica, 16(2), (pp. 227–243).

Miñóna, R., Moreno, L., Martinez, P., Abascal, J. (2014), An approach to the integration of accessibility requirements into a user interface development method. Science of Computer Programming, Volume 86, (pp.58-73) http://dx.doi.org/10.1016/j.scico.2013.04.005

Samp, K., & Decker, S. (2010). Supporting menu design with radial layouts. Proceedings of the International Conference on Advanced Visual Interfaces, (pp.155-162).

Steve, G. & Gangemi, A. & Pisanelli, D. (1997). Integrating Medical Terminologies with ONIONS Methodology.

Zhang, Q.C, Yang, X., Li, P., Huang, G., Feng, S., Shen, C. et al. (2015). Bio-Inspired Engineering of Honeycomb Structure - Using Nature to Inspire Human Innovation. Progress in Materials Science, 74, (pp. 332-400).