Automatic wizard generation

Dániel Szegő

Most state of the art software consists of wizards to assists the user in solving some of her common tasks. Traditionally, these wizards are written by software developers and 'hardcoded' directly into the architecture This usually causes that wizards don't reflect exactly the users' need, since they are predicted during software developing before the software has even finished.

This paper investigates both theoretical and practical ways of writing a software in a way that it would be capable of generating wizards automatically, based on the user's behavior. A framework architecture has been developed and will be briefly introduced, which supports automatic wizard generation and can be added to any software.

One of the key problems of automatic wizard generation relates to nature of generic software systems' architecture. To analyze a software system from the user's point of view, four main layers could be considered. Generation of *components* takes place at the lowermost layer. The *user interface* resides at the top level of the architecture. It usually manifests as a set of buttons, textboxes, links or text items. *Services* can be regarded as an abstraction of calls to components' methods. Actually, when the user has access to the program through the user interface, he manipulates the services. There is a fourth layer between service and user interface, called *glue code*, which consists of event handlers and other useful methods.

Services have been chosen to be the basic steps of a dynamically created wizard, so sequence of services manifest as wizards.

The main task of wizard generator is to create a wizard from user's behavior and embed directly into the software so that it could be applied by the user. Wizard generator consists of two major parts, *model engine*, and *validation engine*, which will be introduced in the followings.

Service calls are collected into a universal sample, while the user is using the software. In other words, all service calls are registered as a sequence of service calls. Wizards should be manifested as common subsequences of all service calls. For example, supposing that user calls *open_connection* and *remote_copy* services many times, a candidate for a wizard could be <*open_connection, remote_copy*> two long sequence. The model engine is responsible for computing common sequences, usually called as *schemas*, from all service calls. Its task is realized by a dedicated algorithm. The algorithm collects all service calls into a universal sample, and finds subsequences of the universal sample which occur at least t times and maximal, in a sense that none of its supersequence occur t times, where t is chosen with the help of heuristics. Three algorithms to compute maximal sequences have been developed (Mschema-0, 1, 1-2) with different time and space complexity. Best of them is Mschema-1 algorithm, which computes maximal sequences with $\mathcal{O}(n^3)$ time and $\mathcal{O}(n)$ space complexity.

Model engine generates common sequence of service calls from the user's behavior, however these sequences cannot be considered directly as wizards because the missing of validation. Validation engine gets wizard candidates from the model engine, and checks the validity of these candidates, with the help of a domain specific formal model. This formal model is based on a relational approach; *must preeced* and *must follow* binary homogenous relations are interpreted between the service calls. A wizard candidate is valid if its service calls are in a well-defined order, in a sense that the order satisfies both relations. The valid candidates become real wizards; the non-valid ones are dropped or transformed into valid (repair). Algorithms for validating and repairing wizard candidates, based on the relations, have been developed and successfully implemented, and added to automatic wizard generator architecture.

An automatic wizard generator architecture, consisting of both the model and validation engine, has been developed and implemented in Java. It can be added to a software, so that the software would support automatic wizard generation. The software should be written with the help of a well-defined design pattern, which supports the dynamic appearance of the user interface. If so, wizard generator classes can be automatically added to the software, without further implementation efforts, and the software will be able to generate wizards automatically.