Code Characterization for Automatic User Interface Generation

Jaroslav Kadlec
Department of Computer Graphics and Multimedia
Faculty of Information Technology, Brno University of Technology
Bozetechova 2
612 66 Brno, Czech Republic
kadlecj@fit.vutbr.cz

***Abstract*—This paper presents extended taxonomy that can be used for better automatic user interface generation. Taxonomy is focused on code characterization allowing an artificial intelligence user interface generator to create user interface which is not limited by visual presentation of characterized data. This allows user interface to contain lots of various operation over characterized data creating rich and interactive user interfaces that could not be created with data characterization only.**

I. Introduction

Creation of user interfaces in various applications is getting more and more complicated. Users request high quality user interfaces and complex applications that are user friendly. Users also expect to have same applications on various devices such as phones, PDAs, notebooks and others. Creation of interface of application that can be portable between various platforms is very difficult, leading in creation of multiple user interfaces which are based on expected device's capabilities and features. Creation of such user interfaces is problematic, leading to increased time of application development. That is why a concept of automatic generation of user interfaces was developed.

Automatic user interface generation systems promise to simplify an application programmer's design tasks by providing a set of design rules [1] and effectiveness criteria [2]. To establish these criteria, it is necessary to understand which of the properties of the information to be visualized are related to user interface design and how they are related. This task is called data characterization [3, 4]. With flexible data characterization it is possible to create automatic presentation systems. These however do not allow creation of rich user interfaces. To create rich user interface with possibilities of various operations over the characterized data, a code characterization is required.

With complete data and code characterization, application programmer is able to describe application code that will be used for automatic user interface generation [5]. Because automatic user interface generation is very complex process requiring artificial intelligence, current user interface management systems are using designer made user interfaces [6, 7, 8]. Such user interface management systems benefit from current code which is independent on user interface which can be simply modified for use with various devices having different presenting capabilities. Also code can be modified independently from user interface design or its components which increases maintainability of application.

II. Previous Work

First data characterization taxonomy was proposed by Mackinlay [2] who was using data properties to guide automatic design of visual presentations. The taxonomy was primarily designed for quantitative data. This taxonomy was later extended by Roth and Matis [3] to address more complex quantitative data. Arens, Hovy and Vossers [9] developed a vocabulary that was able to describe multimedia information. Wahrend and Levis [10] introduced partitioning of data into several categories such as shape and structure. Zhou and Feiner [4] restructured taxonomy into six domains, introducing data role and data sense. While data role characterized data based on user information seeking goals, data sense represented user interpretation preferences.

III. Code Characterization

Code characteristic is expressed using annotation tags that are divided into five dimensions: *command category, command attribute*, *command parameter*, *command sense* and *event*. *Command category* represents set of operations or commands that can be executed with the object. *Command attribute* defines basic properties of the operations and defines various usage options. *Command parameter* describes properties and options for command parameters. *Command sense* distinguishes how should be commands treated. *Events* define optional reactions of user interface on internal object changes.

Code characterization is proposed for object-oriented environment and that is why each piece of information is supposed to be object. Each object belongs to data domain and has data type as defined for data characterization in [4] which

T. Sobh (ed.), Innovations and Advanced Techniques in Computer and Information Sciences and Engineering, 255–259.

is considered to be very complex for data characterization task and which was slightly modified and extended to support code characterization presented here. Each data characterized object can define public methods that can be characterized using code characterization. Because of object-oriented environment, every inherited object inherits public data and methods including data and code characterization. Because taxonomy for data and code characterization is flexible, new types of methods or relations can be easily added to the taxonomy. Following subsections describe code characterization taxonomy in more detail.

A. Command Category

Command category defines set of operations or commands that can be executed on the object. A typical example of command category can be set of methods that allow playing, stopping, pausing of recorded data (might be audio or video playback, or any other data that object can replay or record). Set of such commands have usually default symbolic representation and users are used to this concept from real world or other applications. Command categories can be easily defined in XML format and extend existing set of categories. Categories can be represented in user interface using concept of Smart-Templates [11] which contain information about default symbolic representation and also default graphical design. Because object can be of various categories, multiple categories can be defined for single object. Following categories are basic for most applications and should always be implemented.

1) Collection

Collection contains methods for adding, removing and selecting items contained in object. With this category it is possible to create various objects that act like collections of various data with possible multi-selections and other functions. Add or Remove methods can add or remove objects of various types and do necessary checking before added or removed object is processed. This functionality was almost impossible with data characterization only.

2) Storage

Storage defines methods for saving, opening, and creation of contents. Methods are usually represented by common symbols (e.g. save by diskette icon) and are very important in applications that want to add functionality of creation of new objects and their saving or loading.

3) Navigation

Methods in navigation are most usually used for moving of cursor in collections. Navigation defines actions for previous or next item, first or last item. Navigation can be defined separately from collection, because it can control cursors in various collections at once. A typical example is media player with playlists: user can select one playlist and let play next media file from selected playlist.

4) Media Player

This category defines methods for start (play), stop and pause of playback. This methods are usually represented by standard symbols and should be always implemented in default order.

5) Clipboard

Clipboard is widely used technique for data copying and moving. Clipboard category describes methods that can be used for this purpose and that is why user interface can offer this functionality to user. Typical clipboard methods are copy, paste and cut, storing selected data to clipboard for other applications that may use them.

6) Draggable

Object with method that belongs to draggable category informs that object can be dragged. When object can be dragged, user interface allows user to drag certain objects to another objects. Altough object can be data characterized to be draggable, it is missing method that could process the drag request. This method now can be characterized using code characterization.

7) Droppable

Object containing methods with droppable category contain logic of checking whether dragged object can be dropped and acquiring dropped object. Because drag and drop operations require some logic that cannot be achieved by data characterization, draggable and droppable categories are needed.

B. Command Attribute

Attributes express various information about methods that are implemented by the objects. Although it might seem that information according to each of the method are very different, they have quite lots of common attributes that need to be defined for proper creation of user interface. If some object implements a method that should be visible in user interface, basic attributes such as *name*, *description*, or *importance* should be defined.

1) Name

Name attribute contains name of the method that should be presented to user in user interface. Name is not required for methods that already have its category, because it already has name attribute. Name attribute can also be specified in multiple languages so that user can choose which of the languages he prefers.

2) Description

Description describes method and acts like a tip for the user. When user wants to call certain command from menu or from other part of user interface, system can display or play short help for the command based on the capabilities of the device or user preferences.

3) Importance

Some commands are more important than others. It is good practice to make such commands more accessible. In graphical user interfaces, important commands are placed to toolbars or tool strips so that user can easily access them. Importance attribute represents how important some commands are and whether it should be somehow highlighted or placed in user interface to a location where it is easily accessible.