VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

CHARAKTERIZACE KÓDU PRO AUTOMATICKÉ GENEROVÁNÍ UŽIVATELSKÉHO ROZHRANÍ
CODE CHARACTERIZATION FOR AUTOMATED USER INTERFACE CREATION

AUTOR PRÁCE   ING. JAROSLAV KADLEC
AUTHOR

VEDOUCÍ PRÁCE   DOC. DR. ING. PAVEL ZEMČÍK
SUPERVISOR

OPONENTI   TBD
OPONENTS

DATUM OBHAJOBY   TBD
DEFENSE DATE

CONTENTS

1 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 [3] and effectiveness criteria [11]. 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 [19][21]. 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 [7]. Because automatic user interface generation is very complex process requiring artificial intelligence, current user interface management systems are using designer made user interfaces [4][8][13]. 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.

2 PREVIOUS WORK

2.1 Characterization

First data characterization taxonomy was proposed by Mackinlay [11] 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 [19] to address more complex quantitative data. Arens, Hovy and Vossers [2] developed a vocabulary that was able to describe multimedia information. Wahrend and Levis [20] introduced partitioning of data into several categories such as shape and structure. Zhou and Feiner [21] restructured taxonomy

5

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.

2.2 Generating of user interface

Number of the systems exists from 1980's that use various techniques for generating of user interface. A level of automation provided by these systems varies from the programming abstraction (e.g. UIML [1]) to design tools (e.g. ProcSee [18]), through the mixed systems requiring partial assistance from user interface designer (TERESA [16]). Such systems that provide some mechanisms to automatically generate user interface often use simple rule based approach where every type is matched to the specific user interface element (e.g. UBI [14]). Some systems rely on the type-based declarative model of the information exchanged through the user interface called Abstract User Interface [17]. In many cases, a user interface was specified explicitly (e.g. UIML) or inferred from a code [7]. Some systems include additional information about a high level task or dialogue model (e.g. ConcurTaskTrees [15]) or the task models [5]. Other systems generate the user interface using constraint satisfaction and optimization (e.g. Supple [6]).

3   THESIS OBJECTIVES

The main objective of this thesis is to define approach for automated user interface generation that would be simple to use and allow automated creation of user interface for various platforms.

4   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 a 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 [21] 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

6

the taxonomy. Following subsections describe code characterization taxonomy in more detail.

4.1 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 [12] which contain information about default symbolic representation and also default user interface 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.

4.1.1 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.

4.1.2 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.

4.1.3 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.

4.1.4 Media Player

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

7

4.1.5 **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.

4.1.6 **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. Although 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.

4.1.7 **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.

4.2 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.

4.2.1 **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.

4.2.2 **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.

4.2.3 **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.

4.2.4 **Representation**

Representation defines a symbol that represents certain commands. Representation is especially important for commands that have high importance. Representation can be defined as icon or image. For commands that belong to a category representation is most usually defined in category definition file. In this case, another definition of representation overrides default representation.

4.2.5 **Dependence**

Every method in the code has some dependencies that must be satisfied before the code is executed. This is most usually checked by developer programming user interface. In automatic user interface generation, user interface has to know when is command enabled or disabled. So dependence expresses conditions which have to be satisfied to allow execution of selected command. Dependence is very important because without proper dependence checking, user can have access to unavailable commands. An example can be playing of media file although there is no media file open.

4.3 Command Parameter

Almost every implemented method has some parameters. Parameters init method and method most usually works with available parameters or internal variables. That is why its characterization is very important. Following parameters are basic parameters that are important for proper calling of selected method. An example can be drawing to image. Method that can draw a line has three main parameters: color, starting point and ending point. User interface based on parameter characterization knows that form, which is displaying the image, has a color palette to which is related color in draw line method. User interface will not require user to specify this parameter manually and take the value from the color palette. Because points are data characterized and related to image, user interface knows that the location should be taken from the displayed image and when user selects draw line command, user interface expects user to select two points in the area. After both points are selected, command is executed. Following parameters are important for proper parameter characterization.

4.3.1 **Name**

Describes name of the parameter and is similar to name in command attribute. Name is not always required but is very important in user interfaces that are not based on visual interfaces (e.g. for blind people).

4.3.2 **Description**

Description acts as a tip for the parameter. It is also similar to description in command attribute.

4.3.3 **Relation**

Relation is important for the parameters because it describes relation between parameter and another object in environment. When relation is specified, user interface automatically takes result of relation as input for the parameter. An example is color from color palette. Drawing methods require color which is taken from color palette automatically.

4.3.4 **Default value**

Some parameters, most usually quantitative, require certain value. Although user can change the value, sometimes it is more convenient to show user some kind of the default value that can be used for the command too. An example is number of passes of some computational task. Although user can specify much greater value, algorithm can have good results in five passes. Maximum and minimum values are boundaries which can be specified in data characterization for the parameter data type [21].

4.4 Command Sense

Sense helps to distinguish how are methods executed. By default, user selects command, user interface asks for parameters and method is executed with all specified parameters. Such default sense is called command. When tool sense is specified, user interface runs method again and again as long as the command in context is selected. Typical example is selecting some tool - e.g. a draw line tool - from toolbox. As long as user keeps up with specifying points in the image or as long as the tool is selected, user interface calls draw line command repeatedly and user draws lines. Command and tool sense cover most of the types of method executions and that is why various types of user interfaces can be generated.

For some objects there also exists default action that can be executed whenever user selects the object and launches default action. In graphical user interfaces this is most usually done by double-clicking. For blind users, there is usually voice command run or open. In this case, it is possible to specify which command is default and should be used in such cases.
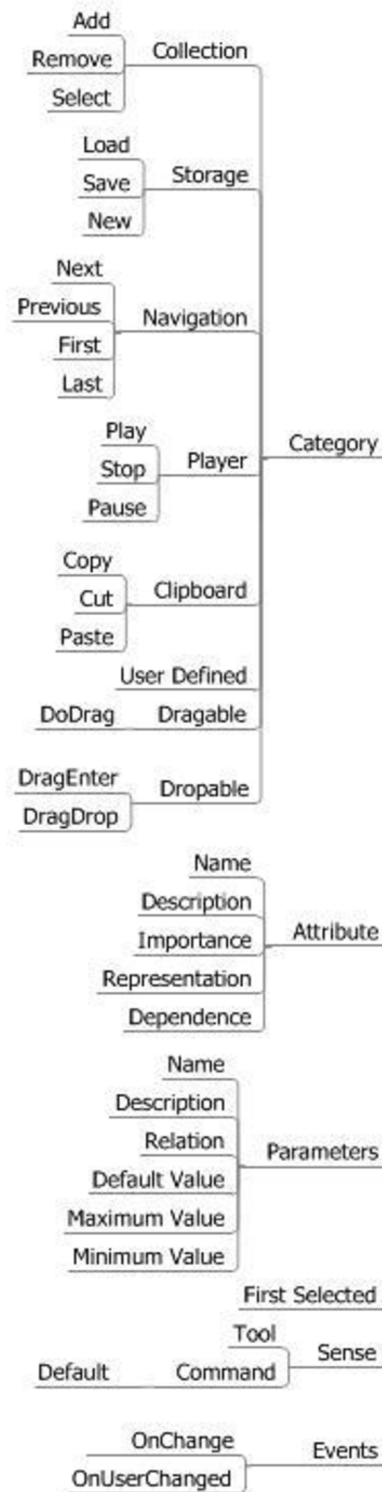
10

Add
Remove — Collection
Select

Load
Save — Storage
New

Next
Previous — Navigation
First
Last

Play
Stop — Player — Category
Pause

Copy
Cut — Clipboard
Paste

User Defined

DoDrag — Dragable

DragEnter — Dropable
DragDrop

Name
Description
Importance — Attribute
Representation
Dependence

Name
Description
Relation
Default Value — Parameters
Maximum Value
Minimum Value

First Selected
Tool — Sense
Default — Command
Events
OnChange
OnUserChanged

Figure 4.1: Code characterization taxonomy.

4.5 Event

Events represent a mechanism that informs user interface for application code that there have been some changes in the data from the user or from the application code. Although various approaches exist for implementing such behavior using callbacks [10], TAPS [3] or ORB [13], event mechanism is very simple and works well for

duplex communication. Base event, that should be implemented for every object is changed. Such event should be executed always when application code changes data of certain object. Because user interface cannot detect changes in data without having checked every available public property, signaling data change using change event is significant to increase performance of application. For data, that have data characterization with high dynamic transience [21], user interface can check its state very often and optimize the performance. Event mechanism can be used for static data or data with low dynamic transience.

Because user interface sets the data that are changed in user interface immediately to the object's properties, it is not necessary to inform object about changes in public data. However for languages that do not support concept of properties (e.g. C++) it is necessary to implement set method that should be characterized by userChanged. This event contains also definition for what data is event designed so that user interface can react correctly when user changes any data.

5 GENERATING USER INTERFACE

The approach is based on data and code characterization described above and is combining rule-based concept for the interface abstraction from the characterized code and constraint satisfaction and optimization for choosing interface objects. General description of this approach was presented in [9]. The first step based on the approach is the code characterization. The characterized code is analyzed and the user interface is automatically created from the analysis, considering the properties of the platform, device, user preferences, and the user context. The next step is a rule based creation of the user interface abstraction using the abstract interface objects. Then, the abstract interface objects are converted to the specific interface objects using the optimization and constraint satisfaction. Finally, the user interface is instantiated. Individual steps of the approach will be demonstrated on a "simple media player" example.

5.1 Data and code characterization

The first step in the approach is the characterization of both the data and code.

```
[DataType(Atomic)]
[DataDomain(Measurement, "Time")]
[DataContinuity(Continuous)]
[DataTransience(Dynamic)]
[DataImportance(0)]
[CodeName("Time Line")]
[CodeDependence("IsMediafileOpened")]
[ParameterRange(0, 0)]
public ulong CurrentPosition{ get; set; }
```

Listing 5.1: Example of the characterized code in C#

12

The data and code characterization is a process of annotation of the data and code with a special tags described above carrying important information for the future user interface generation process. The characterization can be stored in a separate file or directly in the source files (see Listing 5.1) and compiled and linked to the library or assembly.

5.1.1 Example

A possible characterization of selected data and code will be demonstrated on a simple media player. Media player consists of a playlist with media files to play, pause, stop, next, and previous buttons, and a timeline to see and jump into selected part of media file. Media player also has functions allowing adding of media files and directories and removal of selected items from the playlist or clearing the list. Method AddMediaFile(string mediaFile) can be code characterized by Category(Collection(Add, playlist)) so that method is add method for the playlist and user interface can group this list with this command together. Attribute(Name("Add media file")) describes name to present to the user, Attribute(Description("Adds selected media file to playlist.")) is a tooltip for the user, Attribute(Importance(High)) to have the control quickly available. Representation remains the same as specified in category and there is no dependence for the addition of a media file. Parameter(mediaFile, Name("Media File")) defines name of parameter, data characterization consist at least of type(Atomic) because path to file is not divisible, domain(Entity(Path("*.mp3"))) defines that string is a path that should be specified by the user. With this data and code characterization, user interface will show user a default exploring window and allows selection of files with .mp3 extension. AddMediaFile method loads files that were selected by the user from explorer window and launches event OnPlaylistChanged which is characterized as Event(OnChanged(playlist)). This will let user interface update contents of the list. Method that would allow addition of whole directory can be characterized alike except that mask for the Path will be di erent. Play method that starts playing of selected item in the list will be of Category(MediaPlayer(Play)) so there is no need for name, description or representation specification as it is already done in the category. Play is dependent ona method that checks if list contains any items - Attribute(Dependency(IsSelectedItem)). When object changes, this method must be reevaluated to enable or disable the Play command. When user adds items to the list using the Add media file command, the Play button and other buttons that are dependent on IsSelectedItem will be available and user can start playing media files in the list.

5.2 Loading code characteristics

After the code and data characterization is done, the data and code characteristics should be loaded into a characterization tree. The characterization tree reflects well

the structure of the data types and their properties and can be used for the creation of the abstract interface objects. A process of the creation of the characterization tree is shown in Listing 5.2.

```
LoadCharacteristics(tree, dataType)
{
  instance = new Characteristics();
  ParseCharacteristics(instance, dataType);
  foreach(variable in dataType)
  {
    v = LoadCharacteristics(variable, instance);
    instance.Data.Add(v);
  }
  foreach(method in dataType)
  {
    m = LoadCharacteristics(method, instance);
    m.Params = ParseParams(method);
    instance.Code.Add(m);
  }
  tree.Insert(instance);
}
```

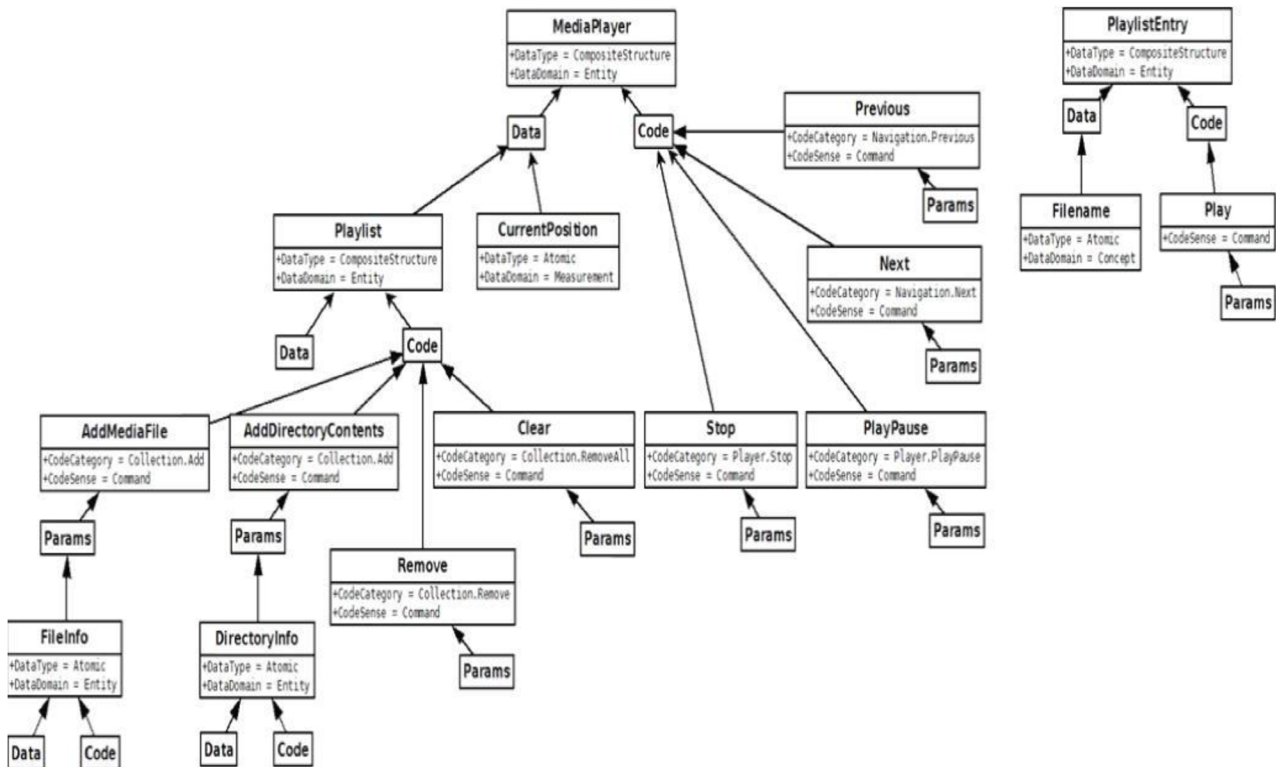*Listing 5.2: Creation of characterization tree*

Figure 5.1: The characterization tree of the simple media player

First, an instance of the structure holding the characterization data is created. Second, attributes are parsed and stored in the structure. Next, every variable is

14

parsed recursively. If the variable has already been parsed, a reference to the node is saved. Similarly, for the methods, every method is parsed including its parameters and stored in the tree. The resulting characterization tree of the media player is shown in the Figure 5.2.

5.3 Creating Abstract Interface Objects

The Abstract Interface Objects (AIOs) are used to represent a user interface structure. They do not represent specific user interface elements but rather indicate what should be the data or code meaning in the terms of the user interface (the data structure can be e.g. represented as a container with a public data). The process of creation is rule-based with domain limitation and takes into consideration user context and preferences. The process of creation of AIOs is presented in the Listing 5.3. AIO database is loaded, including all the rules for every AIO. The characterization tree is parsed and for every node in the tree AIO is picked and initialized. Initialization for every kind of AIO can be different.

```
LoadAIODatabaseFromEepository();
CreateAIOs(char, prefs, context)
{
  selectedAIO = EvalRuleSet(char, prefs, context);
  selectedAIO.Initialize(char, prefs, context);
  char.AIO = selectedAIO;
  foreach(dch in char.Data) {
    CreateAIOs(dch, prefs, context);
    selectedAIO.Add(dch.AIOs);
  }
  foreach(cch in char.Code) {
    if (cch has category && smarttemplate exists) {
      cch.AIO = GetSmartTemplate(cch.category);
      cch.AIO.Link(cch);
    } else {
      cch.AIO = EvalRuleSet(cch, prefs, context);
      if (cch.Params > 0) {
        dlg = CreateDialogAIO();
        foreach(param in cch.Params)
          CreateAIOs(param, prefs, context);
        cch.AIO.Add(dlg);
} } } }
```

Listing 5.3: Creation of AIO tree:

To support a default behavior for commands, a smart template concept [12] is used which groups commands of similar category together. For the methods that

15

require attributes, a dialog AIO is created and filled with AIOs representing each parameter respectively. Figure 5.2 demonstrates the resulting AIO tree created from the characterization tree. The main media player object is represented by a container, playlist as a collection, and seek-bar as a time measurement (both are sub-objects of media player class). The methods were linked together into three main AIOs thanks to smart template. The Playlist entry was placed separately during the collection initialization because it is used for the internal representation of a collection items and is not explicit part of the media player interface.
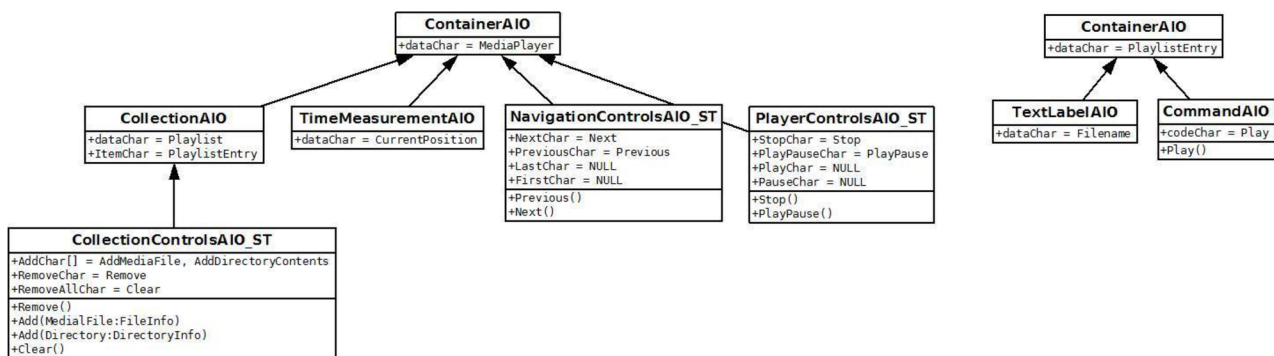

Figure 5.2: AIOs generated from the characterization tree

5.4 Creating specific interface objects

The specific interface objects (CIOs) contain information about the specific user interface element that will be used in the final user interface. The AIO tree with a user interface structure is used to choose the best CIO for every data or code element. The presented process is generally enumeration of all the possible ways of choosing and inserting user interface elements. The best solution with the smallest e ort needed for the interaction is chosen. This process is described in Listing 5.4 and Listing 5.5. The first step is evaluation of a cost function. The cost function evaluates the effort of the user in the interaction with current user interface objects in his current context and specified device. When the current cost is worse than the best solution found so far, conversion will not continue.

The second step is checking if all the AIOs were converted to the CIOs and saving solution. The third step enumerates all the CIOs available for the concrete AIO. Each of these CIOs is applied to the user interface without violating constraints. AIO conversion is repeated for sub AIOs recursively. AIOs with a higher importance are always placed first. Finally, the CIO is removed from the user interface because it can be replaced by other CIO in the last step of previous recursion.

16

```
AIOsToCIOs()
{
  foreach(SubTree in ChTree) {
    while(true) {
      ConvertToCIOs(SubTree, SubTree.AIO, Context, Device);
      if (ConversionComplete(SubTree)) break;
      RegroupLstImportanceContainer(SubTree);
} } }
```

Listing 5.4: Creation of CIO tree.

```
ConvertToCIOs(ChTree, AIO, Context, Device)
{
  if(CurrentCost(ChTree,Context,Device) >= BestCost) return;
  if (AllCIOsApplied()) {
    BestCost = Cost;
    BestCIOs = ChTree.CIOs;
    return
  }
  CIOs = GetCIOs(AIO, Context, Device);
  foreach(CIO in CIOs) {
    if(ApplyCIO(CIO, AIO, Device)) {
      subAIOs = GetSubAIOs(AIO);
      SortByImportance(subAIOs);
      foreach(subAIO in subAIOs) {
        ConvertToCIOs(ChTree, subAIO, Context, Device);
  } } }
  UndoLastCIO();
}
```
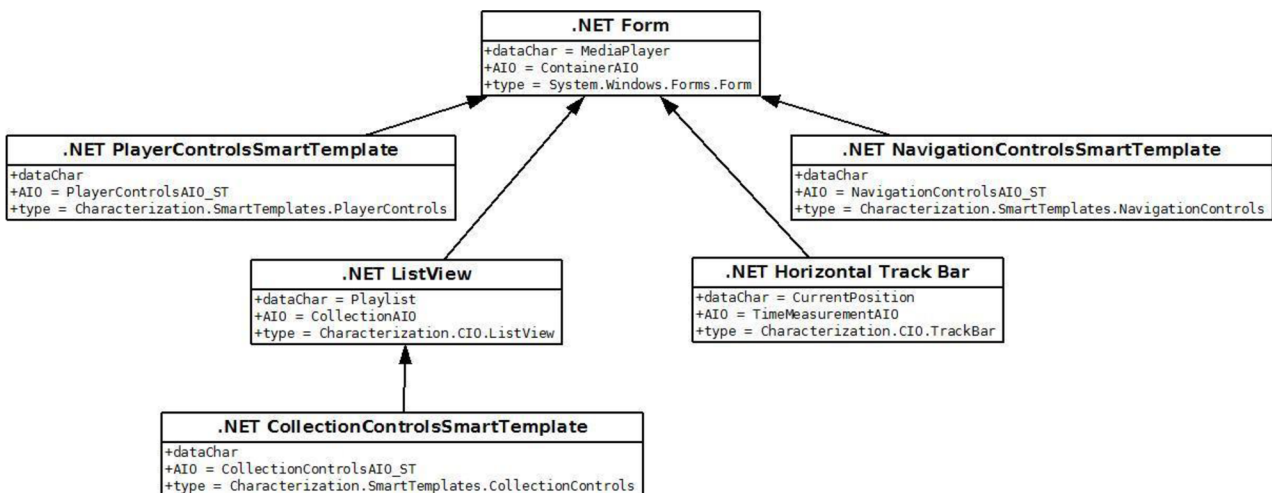
Listing 5.5: Conversion of AIOs to CIOs

Figure 5.3: Created CIOs from the AIO tree.

17

Figure 5.3 demonstrates the result of the conversion to the CIOs. Main class is represented by the Form (window), containing ListView for the playlist, track bar for the seek-bar and the smart templates for categorized commands.

5.5 Instantiating

Instantiation creates instances of CIOs and is responsible for generation and registration of events. The instance of every CIO is created so that instances have the same sub-objects as CIO nodes. Then, the parameters of the CIO are set to the instance. The CIOs representing a data register their dependencies on other objects, events for value changes of the data and the user interface instance. CIOs representing a code register their dependencies and implementing routines calls, generate events to show asterisks and code to show a dialog for input of the parameters if required. Figure 5.4 shows final user interface generated from CIO tree in Figure 5.3. All CIOs were placed in the top-bottom and the left-right order representing highest to lowest importance.
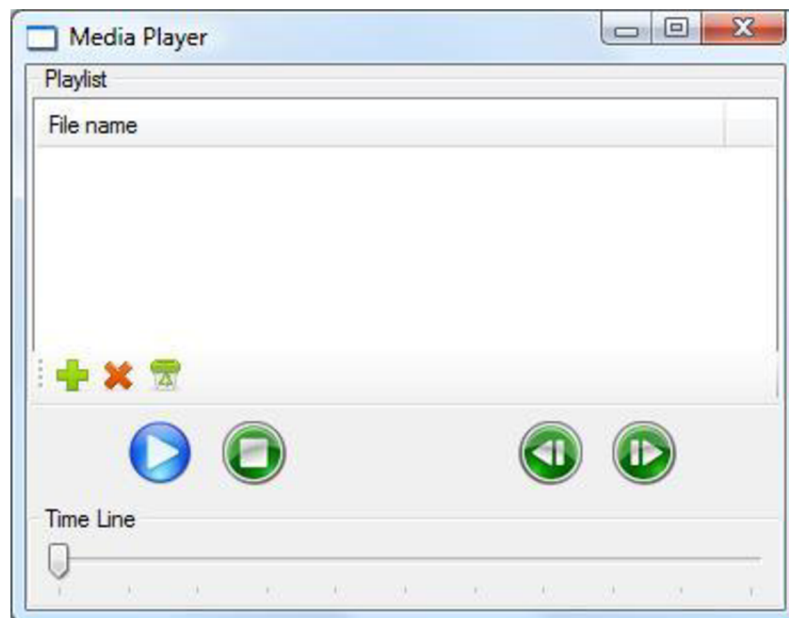

Figure 5.4: Instance of CIOs.

6 CONCLUSION

The goal of this work was to define approach for automated user interface generation that would be simple to use and allow automated creation of user interface for various platforms. This goal was fulfilled.

A taxonomy and processes proposed and described in this work allow new way of application development. While developers were forced to redesign user interface every time a major change in the application has been done, using code characterization together with automated user interface generation process can shorten development time and allow developers to test new methods and functions

18

as the application is being developed even in early stage of development. The proposed taxonomy and its usage lead to benefits in the user interface design. The key benefits of the proposed characterization taxonomy are:

- no expert skills required,
- application code independence on user interface,
- simple application extensibility,
- good maintainability and readability of the application code,
- more effective and faster algorithm development,
- independence on user interface generation process,
- platform independency.

The taxonomy is ideal for creation of user interfaces on multiple platforms. Automated user interface generation process can generate optimal user interface for any device. To proof the concept of the characterization taxonomy, a process of automated user interface generation was presented and demonstrated on examples.

At the moment, the weakest point of the automated user interface generation is generation of layout of user interface elements. An experienced graphics designer is able to produce layouts that are aesthetically superior and more effective than current automatically generated user interfaces. Future work in the problematic of characterization taxonomy should be focused on automated layout techniques to enable higher quality user interfaces, and creation of complex applications to tweak the data and code characterization taxonomy to cover any information required by the automated user interface generation process. Future work should also consider implementation of the user interface generation process for other modalities to take the advantage of the presented approach.

7 REFERENCES

[1] M. F. Ali, M. A. Pérez-quinones, M. Abrams, and E. Shell. Building multi-platform user interfaces with uiml, 2002.

[2] Y. Arens, E. Hovy, and M. Vossers. On the knowledge underlying multimedia presentations. pages 280–306, 1993.

[3] T. Berlage. Using taps to separate the user interface from the application code. In UIST '92: Proceedings of the 5th annual ACM symposium on User interface software and technology, pages 191–198, New York, NY, USA, 1992. ACM.

[4] P. A. Bernstein. Middleware: a model for distributed system services. Commun. ACM, 39(2):86–98, 1996.

[5] F. Bodart, A. marie Hennebert, J. marie Leheureux, I. Provot, and J. V. A model-based approach to presentation: A continuum from task analysis to prototype. 1994.

[6] K. Z. Gajos, J. O. Wobbrock, and D. S. Weld. Improving the performance of motor-impaired users with automatically-generated, ability-based interfaces. In CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human

factors in computing systems, pages 1257–1266, New York, NY, USA, 2008. ACM.

[7] J. Jelinek and P. Slavik. Gui generation from annotated source code. In TAMODIA '04: Proceedings of the 3rd annual conference on Task models and diagrams, pages 129–136, New York, NY, USA, 2004. ACM.

[8] H. Jokstad and S. Carl-victor. Picasso-3 user interface management system, 2002.

[9] J. Kadlec. Steps in automated user interface generation. In Proceedings of Spring Conference on Computer Graphics 2006, volume 22, pages 25–28, 2006.

[10] G. E. Krasner and S. T. Pope. A cookbook for using the model-view controller user interface paradigm in smalltalk-80. J. Object Oriented Program., 1(3):26–49, 1988.

[11] J. Mackinlay. Automating the design of graphical presentations of relational information. pages 66–82, 1999.

[12] J. Nichols, B. A. Myers, and K. Litwack. Improving automatic interface generation with smart templates. In IUI '04: Proceedings of the 9th international conference on Intelligent user interfaces, pages 286–288, New York, NY, USA, 2004. ACM.

[13] K. Nihe, K. Seki, H. Nakamura, R. Yagishita, and T. Shimojima. Distributed object system framework orb, 1994.

[14] S. Nylander, M. Bylund, and A. Waern. The ubiquitous interactor - device independent access to mobile services. CoRR, cs.HC/0305003, 2003.

[15] F. Paterno, C. Mancini, and S. Meniconi. Concurtasktrees: A diagrammatic notation for specifying task models. In INTERACT'97: Proceedings of the IFIP TC13 Interantional Conference on Human-Computer Interaction, pages 362–369. Chapman-Hall, 1997.

[16] F. Paterno, C. Santoro, J. Mantyjarvi, G. Mori, and S. Sansone. Authoring pervasive multimodal user interfaces. Int. J. Web Eng. Technol., 4(2):235–261, 2008.

[17] F. Paterno, C. Santoro, and V. G. Moruzzi. A unified method for designing interactive systems adaptable to mobile and stationary platforms. interacting with. Computers, 15:347–364, 2003.

[18] H. O. Randem, H. Jokstad, T. Linden, H. O. Kvilesjo, S. Rekvin, and A. Hornas. Procsee - the picasso successor, 2005.

[19] S. F. Roth and J. Mattis. Data characterization for intelligent graphics presentation. In CHI '90: Proceedings of the SIGCHI conference on Human factors in computing systems, pages 193–200, New York, NY, USA, 1990. ACM.

[20] S. Wehrend and C. Lewis. A problem-oriented classification of visualization techniques. In VIS '90: Proceedings of the 1st conference on Visualization '90, pages 139–143, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press.

[21] M. X. Zhou and S. K. Feiner. Data characterization for automatically visualizing heterogeneous information. In INFOVIS '96: Proceedings of the 1996 IEEE

8 CURRICULUM VITAE

Education

	1986 – 1991	Basic School Brněnská	Prostějov, CR

	1991 – 1994	Basic School Sídliště Svobody	Prostějov, CR

- Specialized class in light athletic.

	1994 – 1998	High school spec. in electrotechnical engineering	Olomouc, CR

- Specialized in electrotechnical engineering, microprocessor technology.

	1998 – 2003	FIT VUT Brno	Brno, CR

- Ing

Skills

Driving license (type B). Programming in C/C++, Pascal, machine languages, OpenGL, DirectX, .NET. Basics in HTML, DHTML, JavaScript, PHP, SQL. Skilled in work with Microsoft Windows, basics in UNIX based systems.

Interests

Sport, swimming, cycling, martial arts. Music classical, modern. Interested in user interfaces.

Languages

- English (active)
- German (passive, 4 years)
- Spanish (passive, 1 year)

Work

2002 – 2002 (3 months) John Crane Sigma a.s.
Lutín, CR
Developer

- Development of electronical catalogue for central Europe. CD in HTML, multilingual. Contact: martin.grepl@johncrane.cz

2003 – 2005   FIT VUT Brno   Brno, CR
Assistant lecturer

- Computer Graphics Principles, contact: krsek@fit.vutbr.cz
- User Interface Programming, contact: zemcik@fit.vutbr.cz
- Human-Machine Interfaces, contact: zemcik@fit.vutbr.cz

2004 – 2005   Pavel Vavřín   Brno, CR
Programmer

- Development and implementation of algorithms for project Archeological Sights journal (Památky Archeologické) digitizing, containing approx. 41 000 high resolution pages. Contact: pavel_vavrin@yahoo.com

2005 –   VR Group a.s.   Brno, CR
Programmer / Analyst

- Development and implementation of virtual simulators VS-I and VS-II. Project management of Wasp constructive simulation environment. Contact: vit.ryska@vrg.cz

Projects   2004 Augmented Multi-Party Interaction, Brno, CR
Research leader: Hynek Heřmanský
Team Leaders: Burget Lukáš, Černocký Jan, Grézl František, Kadlec Jaroslav, Karafiát Martin,
Matějka Pavel, Motlíček Petr, Potúček Igor, Schwarz Petr, Sumec Stanislav, Španěl Michal, Zemčík
Pavel, http://www.amiproject.org/, Agency: EU-6FP-IST,Code: 506811-AMI

2005 User Interfaces for Hierarchical Structure Visualisation, Brno, CR
Research leader: Kadlec Jaroslav
Team Leaders: Chudy Robert, Zemcik Pavel, Agency: FRVŠ MŠMT, Code: FR200/2005/G1

Publications   Kadlec, Jaroslav: Lip detection in low resolution images, In: Proceeding of the 10th
Conference and Competition STUDENT EEICT 2004, Volume 2, Brno, CZ, 2004, p. 303-
306, ISBN 80-214-2635-7

Herout, Adam; Zemčík, Pavel; Beran, Vítězslav; Kadlec, Jaroslav: Image and Video
Processing Sofware Framework for Fast Application Development, In: Joint
AMI/PASCAL/IM2/M4 workshop, Martigny, CH, 2004, s. 1

Kadlec, Jaroslav; Chudý Robert: Spatial Interface Design, In: ElectronicsLetters.com , Vol.
2004, No. 1, Brno, CZ, p. 13, ISSN 1213-161X

Sumec, S., Kadlec, J.: Event Editor - The Multi-Modal Annotation Tool, In: Workshop on
Multimodal Interaction and Related Machine Learning Algorithms (MLMI), Edinburgh, GB,
2005, p. 1

Kadlec, J., Potúček, I., Sumec, S., Zemčík, P.: Evaluation of Tracking and Recognition
Methods, In: Proceedings of the 11th conference EEICT, Brno, CZ, 2005, p. 617-622, ISBN
80-214-2890-2

Ashby, S., Bourban, S., Carletta, J., Flynn, M., Guillemot, M., Hain, T., Kadlec, J.,
Karaiskos, V., Kraaij, W., Kronenthal, M., lathoud, G., Lincoln, M., Lisowska, A., McCowan,
I., Post, W., Reidsma, D., Wellner, P.: The AMI Meeting Corpus, In: Measuring Behavior
2005 Proceedings Book, Wageningen, NL, 2005, p. 4

Ashby, S., Bourban, S., Carletta, J., Flynn, M., Guillemot, M., Hain, T., Kadlec, J.,
Karaiskos, V., Kraaij, W., Kronenthal, M., lathoud, G., Lincoln, M., Lisowska, A., McCowan,
I., Post, W., Reidsma, D., Wellner, P.: The AMI Meeting Corpus: A Pre-Announcement, In:
Workshop on Multimodal Interaction and Related Machine Learning Algorithms (MLMI),
Edinburgh, GB, 2005, p. 4

Kadlec, J.: Steps In Automated User Interface Generation, In: Proceedings of Spring
Conference on Computer Graphics 2006, Bratislava, 2006, p. 25-28, ISSN 1335-5694

Chudý, R., Kadlec, J.: FOXI - Hierarchical Structure Visualization, In: Proceedings of the
International Conference on Systems, Computing Sciences and Software Engineering
(SCSS05), Bridgeport, US, 2006, p. 5

Kadlec, J.: Code Characterization for Automatic User Interface Generation, In: Innovations
and Advanced Techniques in Computer and Information Sciences and Engineering,
Dordrecht, NL, Springer, 2007, p. 255-260, ISBN 978-1-4020-6267-4

Kadlec, J., Zemčík, P.: Generation of user interface from characterized code, In:
Proceedings of WSCG'10, Plzeň, CZ, ZČU v Plzni, 2010, p. 4

22