

Code Characterization for Automated User Interface Creation

Jaroslav Kadlec^{*}

Department of Computer Graphics and Multimedia
Faculty of Information Technology
Brno University of Technology
Bozetechovat 3, 612 00 Brno, Czech Republic
kadlecj@fit.vutbr.cz

Abstract

Automatic generation of the user interface can simplify development of the computer applications. It can help in the development for various target platforms or in simpler testing and algorithm debugging as the user interface can be created for the actual code and platform taking into account many properties. Process of the automatic generation of the user interface can be supported by the data and code characterization. In this article, an innovative approach using the data and code characterization is presented. The characterization taxonomy, the mechanisms, and algorithms describing how the data and code characteristics are loaded, the way how objects are transformed into abstract and specific user interface elements, and the process of finalizing user interface is briefly described. As an example, simple media player is described in every step of the user interface generation process.

Categories and Subject Descriptors

H.5.2 [User Interfaces]: Automatic creation; D.2.0 [Software Engineering]: Source code characterization

Keywords

user interface, data characterization, code characterization, automated user interface creation

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 into six domains, introducing data role

^{*}Recommended by thesis supervisor:

Assoc. Prof. Pavel Zemcik
Defended at Faculty of Information Technology, Brno University of Technology on January 5, 2011.

© Copyright 2011. All rights reserved. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from STU Press, Vazovova 5, 811 07 Bratislava, Slovakia.

Kadlec, J. Code Characterization for Automated User Interface Creation. Information Sciences and Technologies Bulletin of the ACM Slovakia, Vol. 3, No. 1 (2011) 1-8

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 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. This methods are usually represented by standard symbols and should be always implemented in default order.

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 drop-
pable 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 is representation 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 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.

4.5 Event

Events represent a mechanism that informs user interface or 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 set 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. 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 a directly in the source files (see listing 1) and compiled and linked to the library or assembly.

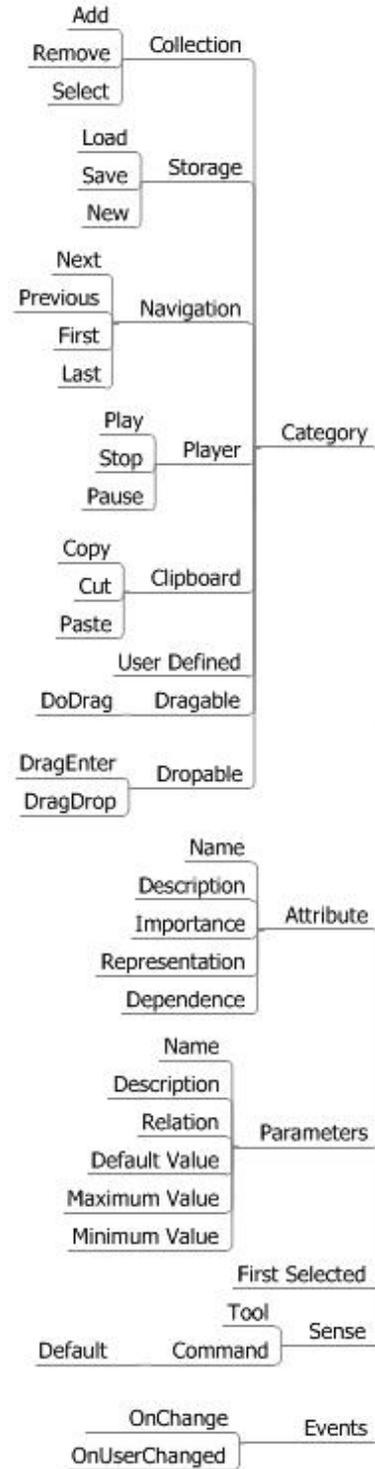


Figure 1: Code characterization taxonomy.

Listing 1: Example of a characterized code in C#

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

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, 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 on 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 different. 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 on a 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 2.

Listing 2: Creation of characterization tree

```
1 LoadCharacteristics(tree, dataType)
2 {
3     instance = new Characteristics();
4     ParseCharacteristics(instance, dataType);
5     foreach(variable in dataType)
6     {
7         v=LoadCharacteristics(variable, instance);
8         instance.Data.Add(v);
9     }
10    foreach(method in dataType)
11    {
12        m = LoadCharacteristics(method, instance);
13        m.Params = ParseParams(method);
14        instance.Code.Add(m);
15    }
16    tree.Insert(instance);
17 }
```

First, an instance of the structure holding the characterization data is created (3). Second, attributes are parsed and stored in the structure (4).

Next, every variable is parsed recursively (7). If the variable has already been parsed, a reference to the node is saved. Similarly, for the methods, every method is parsed (12) including its parameters and stored in the tree (16). The resulting characterization tree of the media player is shown in the figure 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 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 (4) and initialized (5). Initialization for every kind of AIO can be different.

Listing 3: Creation of AIO tree

```
1 LoadAIODatabaseFromEepository();
2 CreateAIOs(char, prefs, context)
3 {
4     selectedAIO=EvalRuleSet(char, prefs, context);
5     selectedAIO.Initialize(char, prefs, context);
6     char.AIO = selectedAIO;
7     foreach(dch in char.Data) {
8         CreateAIOs(dch, prefs, context);
9         selectedAIO.Add(dch.AIOs);
10    }
11    foreach(cch in char.Code) {
12        if(cch has category &&
13            smart template exists) {
14            cch.AIO=GetSmartTemplate(cch.category);
15            cch.AIO.Link(cch);
16        } else {
17            cch.AIO=EvalRuleSet(cch, prefs, context);
18            if (cch.Params > 0)
19                dlg=CreateDialogAIO();
20            foreach(param in cch.Params)
21                CreateAIOs(param, prefs, context);
22        }
23        cch.AIO.Add(dlg);
24    }
```

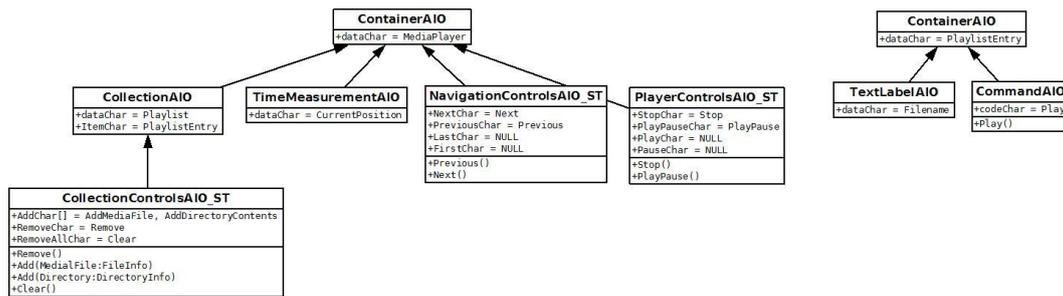



Figure 3: AIOs generated from the characterization tree

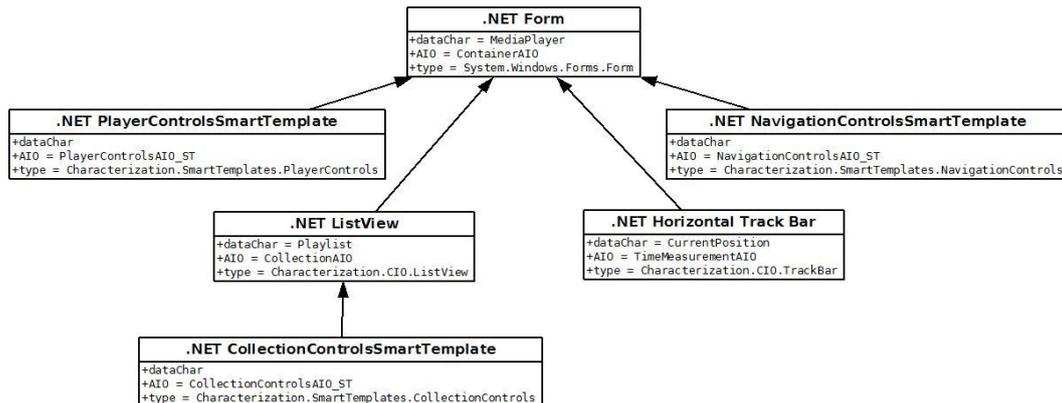


Figure 4: Created CIOs from the AIO tree

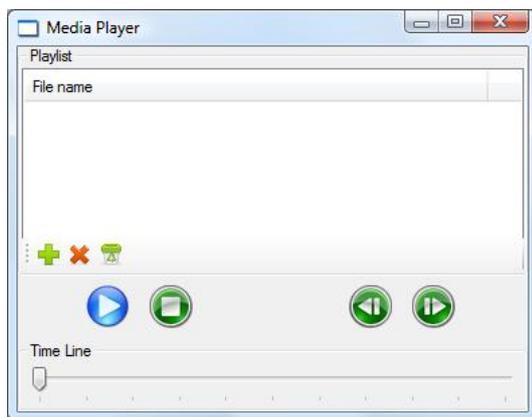


Figure 5: Created CIOs from the AIO tree

Figure 5 shows final user interface generated from CIO tree in figure 4. All CIOs were placed in the top-bottom and the left-right order representing highest to lowest importance.

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 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.

Acknowledgements. This work has been supported by research grant *User Interface with Hierarchical Structures* from FRVS MSMT, *Augmented Multi-party Interaction* from EU-HLT.

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 Jean-marie, L. J. Vanderdonck, 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 International 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 Symposium on Information Visualization (INFOVIS '96)*, page 13, Washington, DC, USA, 1996. IEEE Computer Society.

Selected Papers by the Author

- Kadlec, J., Zeměík, P. Generation of user interface from characterized code, In *Proceedings of WSCG'10*, pages 4, Plzen, CZ, 2010
- Kadlec, J. Code Characterization for Automatic User Interface Generation In *Innovations and Advanced Techniques in Computer and Information Sciences and Engineering*, pages 255–260, Dordrecht, NL, Springer, 2007
- Chudý, R., Kadlec, J. FOXI - Hierarchical Structure Visualization In *Proceedings of the International Conference on Systems, Computing Sciences and Software Engineering (SCSS05)*, pages 5, Bridgeport, US, 2006
- Kadlec, J. Steps In Automated User Interface Generation In *Proceedings of Spring Conference on Computer Graphics 2006*, pages 25–28, Bratislava, 2006
- Ashby, S., Bourban, S., Carletta, J., Flynn, M., Guillemot, M., Hain, T., Kadlec, J., Karaiskos, V., Kraaij, W., Kronenthal, M., Iathoud, 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)*, pages 4, Edinburgh, GB, 2005
- Ashby, S., Bourban, S., Carletta, J., Flynn, M., Guillemot, M., Hain, T., Kadlec, J., Karaiskos, V., Kraaij, W., Kronenthal, M., Iathoud, G., Lincoln, M., Lisowska, A., McCowan, I., Post, W., Reidsma, D., Wellner, P. The AMI Meeting Corpus In *Measuring Behavior 2005 Proceedings Book*, pages 4, Wageningen, NL, 2005
- Kadlec, J., Potůček, I., Sumec, S., Zeměík, P. Evaluation of Tracking and Recognition Methods In *Proceedings of the 11th conference EEICT*, pages 617–622, Brno, CZ, 2005
- Sumec, S., Kadlec, J. Event Editor - The Multi-Modal Annotation Tool In *Workshop on Multimodal Interaction and Related Machine Learning Algorithms (MLMI)*, pages 1, Edinburgh, GB, 2005
- Herout, A., Zeměík, P., Beran, V., Kadlec, J. Image and Video Processing Software Framework for Fast Application Development In *Joint AMI/PASCAL/IM2/M4 workshop*, pages 1, Martigny, CH, IDIAP, 2004
- Kadlec, J., Chudý, R. Spatial Interface Design In *ElectronicsLetters.com*, pages 1–13, Brno, CZ, 2004
- Kadlec, J. Lip detection in low resolution images In *Proceeding of the 10th Conference and Competition STUDENT EEICT 2004*, Volume 2, pages 303–306, Brno, CZ, 2004