PRACE NAUKOWE AKADEMII EKONOMICZNEJ WE WROCŁAWIU
Nr 1081 — 2005
Nowoczesne technologie informacyjne w zarządzaniu

Bożena Jankowska

USER INTERFACE DESCRIPTION LANGUAGES FOR CONTENT GENERATION ON HETEROGENOUS DEVICES

1. Introduction

The advances in wireless networking (in particular deployment of the so-called Third Generation Networks), emergence of powerful mobile devices and the increasing users' requirements towards the availability of information are currently the driving forces behind ubiquitous computing, i.e. providing access to data and computing power independently of devices and locations. The success of ubiquitous computing technologies does not, however, rely exclusively on the improved capabilities of devices or mobile networks but requires also a radical shift in the process of application design. Declarative User Interface Description Languages (UIDLs) which are a relatively old idea, previously well-known and applied mainly in the academic research [Szekely et al. 1995], may provide valuable help in this respect. With the proliferation of heterogeneous devices, the UIDL concept was suddenly revived and applied in many approaches for device--independent content adaptation.

This paper introduces and evaluates the most popular User Interfaces Description Languages for device-independent content generation and is organized as follows: Section two outlines the theoretical foundations of the User Interface Description Languages concept. In section three an overview of the most important UIDLs is provided. In subsequent section the described UIDLs are compared with regard to the features relevant for device-independent approaches. Conclusions are driven in the last section.

2. Concept of User Interface Description Languages

Development of applications for heterogeneous devices implies a considerable effort since similar data have to be presented in a different, device-independent way. In automated content adaptation approaches, the elements of a device

-independent description language are mapped to appropriate building blocks of a device-specific description language. The mapping can follow the intersection approach or the generic language approach.

In the intersection approach, the characteristics of the device-independent markup language are restricted to the common features supported by all devices. Elements of the device-independent language are translated into concrete, device -specific counterparts and change as the underlying device-specific markups are modified. In a generic approach, the device-independent format can encompass characteristics not featured by all device-specific markups. The mapping to a particular representation may be loose and the features of the generic language are not restricted to those applied in device specific language with the "lowest common denominator" capabilities [cf. Göbel et al. 2001].

In the UI development paradigm named model-based interface development a generic interface is constructed using a high-level specification language. A generic user interface is "an interface whose aspects may vary in different devices while its functionality prevails in any of them" [Mayora-Ibarra 2002, p. 2]. The interface is automatically rendered according to device characteristics. Such high -level languages are called User Interface Description Languages (UIDL) and express diverse aspects of the user interface, including its abstract and concrete elements, the tasks to be performed by the user, and the user interface dialogue.

User Interface Description Languages are based on declarative models. A declarative model is defined as "a common representation that tools can reason about, enabling the construction of tools that automate various aspects of interface design, that assist system builders in the creation of the model, that automatically provide context sensitive help and other run-time assistance to users" [Szekely et al. 1995, p. 120]. Well-known models include data/domain models, application models, task models, dialog models, presentation models and user models. In a model, information is usually categorized into three levels of abstraction [cf. Szekely 1996]. The highest level encompasses task, domain and user models. The task model describes the tasks to be accomplished by the user, while the data or domain model provides a description of the objects the user manipulates and of the supported operations. The user model provides information about a user or a group of users.

The second and third levels of the model are responsible for presentation. The second level corresponds to an abstract user interface and specifies the information that will be shown in a window together with a dialog responsible for the interaction with information (dialog model). The abstract user interface consists of abstract interaction objects (AIO), information elements and presentation units[1]. AIOs correspond to interface tasks such as selecting one element from a set, or

[1] This classification is taken from [Szekely 1996]. Other approaches do not use this divison of the abstract presentation model.

showing a presentation unit. Information elements represent information to be shown, in form of constant values (e.g. label), or sets of objects and attributes drawn from the domain model. Presentation units are an abstraction of windows and describe a collection of AIOs and information elements displayed to a user.

The third level of the model, the concrete or final user interface specification, denotes the style for displaying the presentation units, the AIOs and included information elements as well as the layout of elements. It corresponds to the interface in terms of toolkit primitives such as windows, buttons or checkboxes, and graphical primitives such as lines or images.

UIDLs may have different levels of abstractions. The instance level means that the user interfaces are runnable; the model level signifies that one or many models are involved in the development of UIs. If a language specifies the models and their semantics, it is at the meta-model level. The meta-meta-model level is achieved, if for the meta-model level interfaces the fundamental concepts about meta-model development are also provided [Souchon, Vanderdonckt 2003, p. 387].

2. Overview of existing User Interfaces Markup Languages for heterogeneous devices

User Interfaces Markup Languages enjoy growing popularity and their number is continuously increasing. Some interesting languages not described here include USer Interface eXtensible Markup Languge (USIXML) [Limbourg Q. et al. 2004], eXtensible User Interface Language (XUL) from Mozilla[2], Alternate Abstract Interface Markup Language (AAIML) [Zimmermann et al. 2002], SEESCOA XML [Luyten, Coninx 2001] and Abstract User Interface Markup Language (AUIML) from IBM [Azevedo et al. 2000]. However, due to their advantages certain languages are more often used than others and are therefore described in more detail in the subsequent sections.

User Interface Markup Language (UIML). User Interface Markup Language (UIML) [Ali et al. 2004] is one of the most popular approaches for delivering information to different devices in a device-independent way. It is based on XML, provides a declarative description of user interfaces and specifies a canonical format for multiple devices. UIML was designed to separate user interface code from application logic code, to facilitate the reuse of code and to make rapid prototyping of user interfaces for multiple devices possible. Currently, UIML is being standardized by the Organization for the Advancement of Structured Information Standards (OASIS)[3]. An UIML document can consist of seven main elements: <interface>, <structure>, <content>, <behavior>, <style>, <template> and <peers>. The <interface> element embraces all other tags and represents the

[2] Cf. http://www.mozilla.org/projects/xul/

[3] Cf. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uiml.

user interface. In the <structure> element the physical organization of the interface and the relationships between UI elements within the interface are defined. The <content> elements enclose the content of a document (e.g. text, images), separating it from the UI structure. The <behavior> elements describe the behavior of the interface by specifying conditions (e.g. occurrence of an event) and actions associated with them. The <style> elements denote the presentation style of UI elements and the <peers> elements associate widgets, methods, programs or objects in the application logic with the user interface, combining application presentation with its logic. The <template> elements help to describe those parts of UI which are reusable [cf. Harmonia 2002 for the complete specification].

UIML document can be mapped to any type of user interface (e.g. Java AWT, WML, VoiceXML, HTML) with help of appropriate renderers[4]. Most of the UIML renderers are commercial software but the language specification is freely available for public use and can be extended with additional vocabularies. For example in the MObile multimodal Next-generation Applications (MONA) project specific vocabulary for multimodal interfaces and a suitable renderer for graphical and voice user interfaces was developed [Simon et al. 2004].

eXtensible Interface Markup Language (XIML). eXtensible Interface Markup Language (XIML) is another UIDL developed to provide a common representation of multiple user interfaces [Eisenstein et al. 2000]. The language is "an organized collection of interface elements that are categorized into one or more major interface components" [Eisenstein et al. 2000]. XIML contains components, relations, and attributes, whereby relations and attributes can be in form of statements or definitions. Five basic components can be distinguished in the language specification: task, domain, user, presentation and dialog. The task component supports a definition of business processes and user tasks. The domain component represents a collection of data objects and classes of objects structured in a hierarchy. The user component specifies a hierarchy of users. The presentation component defines a hierarchy of interface elements such as window, button, etc. The dialog component describes a collection of elements determining user actions associated with particular interface components. A relation in XIML is described as a definition or a statement that connects two or more elements within one component or across many components. Attributes are features or properties of elements.

XIML was used in the MANNA (Map Annotations Assistant) project [Eisenstein et al. 2000] for creation of multiple user interfaces for annotated maps of geographical areas. In this work, additional concepts for better adaptation to mobile devices were introduced: Abstract Interaction Objects (AIOs), Concrete Interaction Objects (CIOs), Logical Window (LW) and Presentation Unit (PU). An interaction object (also called a widget) is any element that helps to visualize or

[4] Most of the renderers were developed by Harmonia company (cf. [http://www.harmonia.com]).

manipulate information or to perform a task [Eisenstein et al. 2000]. AIOs are elements that cannot be executed on any platform and do not provide implementation details. CIOs are executable components and can be mapped to the platform on which they should run. CIOs are children of AIO; they inherit its properties and give information about implementation details. A Logical Window is a group of simple or composite AIOs (e.g. a window, a sub-window, a dialog box, a listbox) and is itself a composite AIO. A Presentation Unit is a complete presentation environment for enabling an interactive task and can consist of one or many Logical Windows displayed simultaneously or one after another. This presentation hierarchy can be used in a generation of platform-specific presentations from platform independent presentation models.

Dialog Description Language (DDL). Dialog Description Language (DDL) is an XML-based, device-independent markup language which describes a structure of abstract elements [Hübsch et al. 2003]. The root <ddl> element of a document may contain different elements such as: <include> for integrating external source code, <DataTypeDef> for the definition of data types used for validation of user input, <DataInstance> for specifying data instances for user input, <dialog> for the definition of the dialog structure and <part> for modeling the structure of the dialog. Furthermore, different classes can be assigned to various parts. A class is defined as a set of <properties> (styles for presentation or abstract properties). The content of DDL dialogs is enclosed in <content> tag. A data item can be defined using <constant> elements.

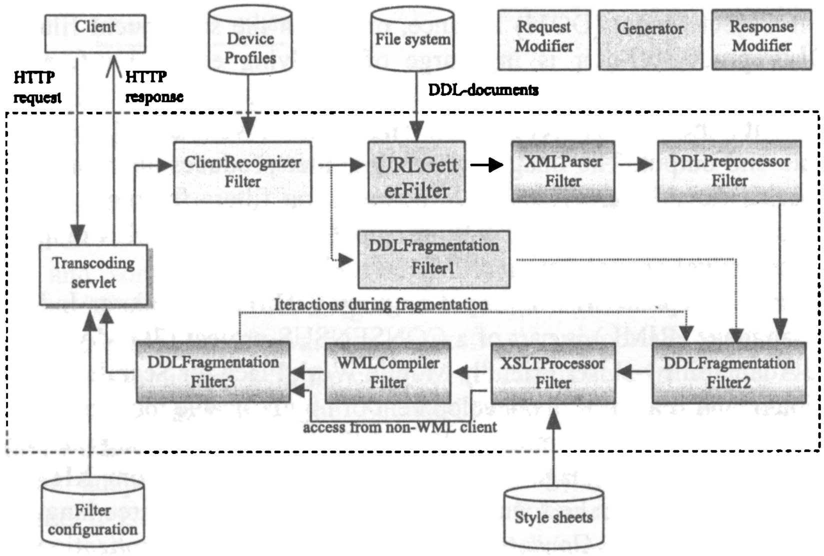

Fig. 1. DDL adaptation framework

Source: [Buchholz et al. 2002, p. 47]

The semantics of the properties is defined separately in a Document Type Description (DTD) [W3C 2004] and can be extended. The container and the source elements are particularly interesting; the remaining elements simply map to traditional Web-based UI elements such as labels, textboxes, frames, forms, etc. The container enables grouping of parts and specification of layout for them. The source element enables the inclusion of non-interpreted (not DDL) device specific source code (e.g. WML).

In order to apply DDL an adaptation framework displayed in Figure 1 was developed. The framework is based on a chain of filters[5] which perform the adaptation according to the DDL specification. Three different types of filters were applied: Request Modifiers, Generators and response Modifiers. The Request Modifiers are processed at first and alter the HTTP request. The Generators supply the requested content and the Response Modifiers transform the retrieved content to the device. The sequence of filters is specified in a configuration file and a filter may also determine its successor.

The most important filters implemented in this adaptation approach are: ClientRecognizerFilter, URLGetterFilter, ServletRunnerFilter, XMLParserFilter, DDLPreprocessorFilter, XSLTProcessorFilter, ImageTranscodingFilter, DDL fragmentation filters and WMLCompilerFilter. The ClientRecognizerFilter is responsible for the recognition of devices according to the User Agent String. The URLGetterFilter retrieves a file and the ServletRunnerFilter invokes an external servlet on the server. The XMLParserFilter convert the DDL document into Document Object Model (DOM) instance, on which the subsequent filters work. The DDLPreprocessorFilter is in charge of resolving external references and inheritance hierarchies. It produces a simplified DDL document which is then processed with the XSLTProcesorFilter and XSLT [W3C 2003] style sheet to appropriate end output. The ImageTranscodingFilter produces appropriate images that fit device capabilities. The DDL fragmentation filters fragment the dialogs, perform user input validation and store input data. The WMLCompilerFilter compiles the textual representation of WML into binary format.

Renderer Independent Markup Language (RIML). Renderer Independent Markup Language (RIML) is part of a CONSENSUS project (3G Mobile Context Sensitive Adaptability – User Friendly Mobile Work Place for Seamless Enterprise Applications), and it aims at the development of highly-usable mobile applications [Ziegert et al. 2004]. RIML combines elements from already existing markups with new elements. In this language borrowed tags and concepts from XHTML 2.0, XForms 1.0 and SMIL can be found [cf. Consensus 2004]. All three languages are recommended by the W3C Consortium for device-independent applications.

[5] As a part of Java Servlet specification the conept of filters was introduced. A filter is able to intercept requests and responses to transform or use the information contained in the requests or responses.

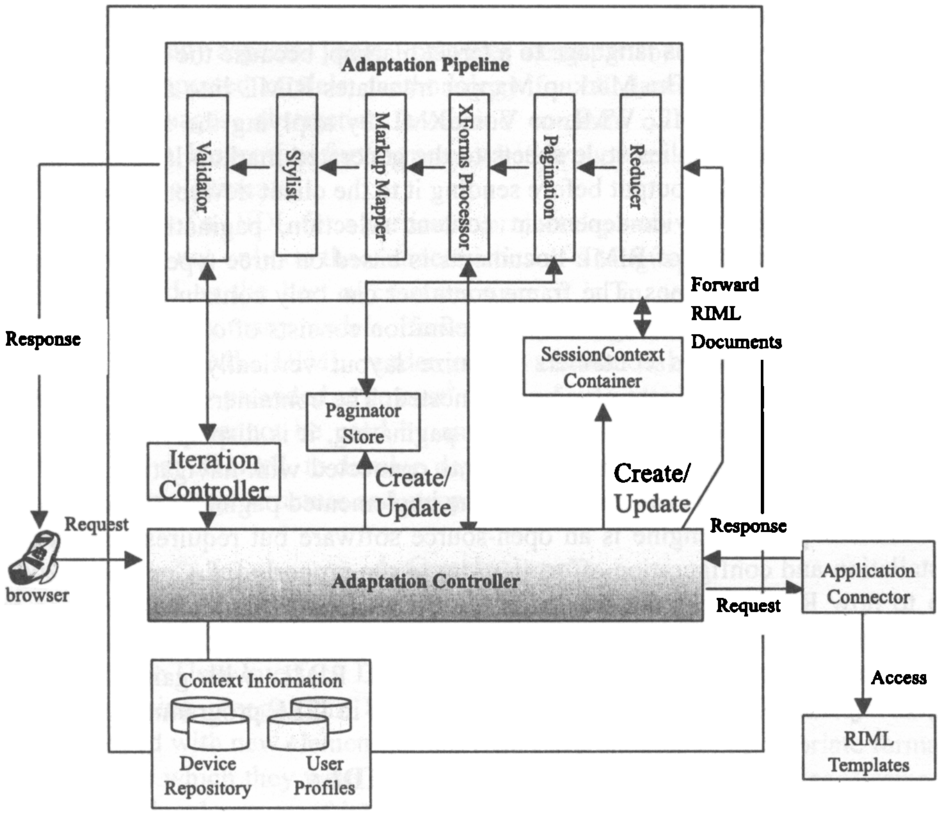

Fig. 2. The architecture of RIML adaptation engine
Source: [Consensus 2004].

The Adaptation Engine (AE) depicted in Figure 2 is used for converting RIML to appropriate formats and consists of an Adaptation Controller, an Adaptation Pipeline and some modules supporting session and context information (cf. [Consensus 2004, pp. 98-100; Ziegert et al. 04]). The Controller is in charge of request processing and forwarding requests to appropriate components. The Controller communicates with the components responsible for storing Device and User Information and with the SessionContext Container to determine the appropriate adaptation for a particular device

In the adaptation process, the Adaptation Pipeline is the most relevant component. It is composed of six elements: a Reducer, a Paginator, an XForm Processor, a Markup Mapper, a Stylist and a Validator. The Reducer selects the content to be displayed basing on the device characteristics. The Paginator is in charge of splitting content into smaller units displayable on one screen and generation of navigation links. All generated pages are stored in the Pagination

Store, from which all the pages are retrieved. The XForm Processor converts the forms written in XForms language to a target markup, because the browsers do not support this standard. The Markup Mapper translates RIML into appropriate end-format: HTML, XHTML, WML or VoiceXML by applying the suitable marker. The Stylist module applies style sheets to the generated markup language and the Validator validates the output before sending it to the client device.

RIML handles device-dependent content selection, pagination, layout and navigation. The layout of RIML documents is based on three types of containers: rows, columns and frames. The frame container can only contain content and not additional layout elements. Each layout definition consists of one or many frames. The columns and rows containers organize layout vertically and horizontally, respectively. The containers may be also nested. The containers can be paginating or non-paginating. If the container is non-paginating, it is displayed on one page; otherwise it is split into multiple fragments connected with navigation links. The pagination is accomplished according to the implemented pagination algorithm.

The Adaptation Engine is an open-source software but requires considerable installation and configuration effort in order to run properly [cf. Consensus 2003]. Up to now RIML lacks any available development environment, facilitating the implementation process, although such support was announced. Without suitable development tools it cannot be expected that RIML will gain widespread popularity and will be used by users inexperienced in J2EE programming.

3. Comparison of UIDLs

Table 1. Comparison of UIDLs

	Models	Methodology	Tools	Supported languages	Open-source	Target/ Abstraction level
UIML	Presentation and dialog models, partially domain model	Specification of multiple UI presentations, factoring/ corrections	Multiple rendering engines, code generator, editor	C++, Java, VoiceXML, HTML, WML, PalmOS, .NET	No	Multi-platform Model level
DDL	Domain and presentation models	Specification of multiple UI presentations	Adaptation engine	WML, XHTML, HTML	No	Multi-platform Model level
RIML	Domain and presentation models	Specification of multiple UI presentations	Adaptation engine	WML, XHTML, HTML, VoiceXML	Yes	Multi-platform Model level
XIML	Task, domain, user, dialog, presentation models	Specification of multiple UI descriptions or generic description of UI	Rendering engine, code editor	HTML, WML, Java	Yes	Multi-platform, context-sensitive applications Meta-model-level

Source: based on [Souchon, Vanderdonckt 2003; Van den Bergh 2004].

User Interface Description Languages may seem similar at the first sight but they differ in many aspects. Table 1 provides a comparison of the described UIDLs in terms of supported models, methodology for UI description, available tools, supported languages, abstraction level and contexts of use (user/environment/ platform model). It specifies furthermore, whether they are open-source products or commercial developments.

The choice of a UIDL for a particular project depends on the goals pursued by this project and not only on the characteristics of a language. For example, XIML can be regarded as the best alternative since it is an open-source, meta-model language supporting many models. However, it lacks good tools' support compared with UIML or RIML. UIML renderers are commercial software while RIML adaptation engine is provided at no cost[6]. The main disadvantage of all languages is the fact that they cannot be used by an average user with the knowledge about HTML, WML or XHTML to develop device-independent presentations because of complicated language structure and steep learning curve.

4. Conclusions

With the proliferation of mobile devices server-side adaptation approaches in general and the User Interface Description Languages (UIDLs) in particular have gained increasing recognition. UIDLs describe generic User Interfaces, can be easily extended with new elements and can be then rendered to appropriate formats for devices for which they were not initially developed. These advantages came at a price – the development of interfaces in such languages is usually difficult and requires knowledge of a particular syntax and of the language's peculiarities. Since most of the developed UIDLs do not offer any programming tools, future research and development should particularly focus on Integrated Development Environments for these languages. Otherwise, they will share the fate of the old UIDLs and will remain a powerful concept used mainly by academic researchers. This is especially important for RIML which offers a rich functionality but is still not applied by mobile content authors because of its complexity and lacking IDE support.

References

Ali M., Pérez-Quiñones M., Abrams M.: *Building Multi-Platform User Interfaces with UIML*, [in:] *Multiple User Interfaces: Cross-Platform Applications and Context-Aware Interfaces*, eds. A. Seffah, H. Javahery H.,Wiley & Sons, Sussex, 2004, pp. 95-118.

[6] See http://sourceforge.net/projects/consensus.

Azevedo P., Merrick, R., Roberts, D., OVID to AUIML - User-Oriented Interface Modelling, in: Proceedings of the 1st Workshop Towards a UML Profile for Interactive Systems Development (TUPIS'00), York, 2000. http://math.uma.pt/tupis00/submissions/ azevedoroberts/azevedo-roberts.html. Download: 2005-01-02.

Buchholz S., Göbel S., Ziegert T., Schill A., *Software Architecture for the Adaptation of Dialogs and Contents to Different Devices*, "Proceedings of the International Conference on Information Networking, Wireless Communications Technologies and Network Applications", Cheju Island, 2002, pp. 42-51.

Consensus, *Adaptation Engine: Installation and Configuration Guide*, 2003. http://www.consensus-online.org/publicdocs/D7public.pdf. Download: 2005-04-10.

Consensus, *RIML Language Specification*, Version 2, 2004. http://www.consensus-online.org/publicdocs/20040317-RIML-II-Final-public.pdf. Download: 2005-04-01.

Eisenstein J., Vanderdonckt J., Puerta A.: *Adapting to Multiple Contexts with User-Interface Modeling*, "Proceedings of the 3rd IEEE Workshop on Mobile Computing Systems and Applications", Monterey, 2000, pp. 83-94.

Göbel S., Buchholz S., Ziegert T., Schill A., *Device Independent Representation of Web-based Dialog and Contents*, "Proceedings of the IEEE Youth Forum in Computer Science and Engineering (YUFORIC'01)", Valencia, 2001. http://www.rn.inf.tu-dresden.de/scripts_lsrn/veroeffent_print/YUFORIC2001.pdf. Download: 2005-04-12.

Harmonia, *User Interface Markup Language (UIML) Specification*, Version 3.0, 2002. http://www.uiml.org/specs/. Download: 2005-04-12.

Hübsch G., Springer T., Schill A., Spriesterbach A., Ziegert T., *Systemlösungen für die Entwicklung adaptiver Anwendungen für mobile und ubiquitäre Infrastrukturen*, "HMD, Praxis der Wirtschaftsinformatik", Heft 229, 2003, pp. 42-55.

Limbourg Q. et al., *USIXML: a Language Supporting Multi-Path Development of User Interfaces*, "Proceedings of the 9th IFIP Working Conference on Engineering for Human-Computer Interaction", Hamburg, 2004, pp. 89-107.

Luyten K., Coninx K., *An XML-based Runtime User Interface Description Language for Mobile Computing Devices*, "Proceedings of the 8th International Workshop on Interactive Systems: Design, Specification and Verification", Glasgow, 2001, pp. 17-29.

Mayora-Ibarra O., *Generation of Device-Independent User Interfaces*, "Proceedings of the International Workshop on Research & Development of Human Communication Technologies for Conversational Interaction and Learning", Puebla, 2002, pp. 1-3.

Simon R., Jank M., Wegscheider F., *A Generic UIML Vocabulary for Device- and Modality Independent User Interfaces*, "Proceedings of the 13thInternational Conference on World Wide Web", New York, 2004, pp. 434-435.

Souchon N., Vanderdonckt J., *A Review of XML-compliant User Interface Description Languages*, "Proceedings of the 10th International Conference on Design, Specification, and Verification of Interactive Systems", Madeira, 2003, pp. 377-391.

Szekely P. et al., *Declarative Interface Models for User Interface Construction* Tools: the MASTERMIND Approach, "Proceedings of the 7th Working Conference on Engineering for Human-Computer Interaction", Yellowstone Park, 1995, pp. 120-150.

Szekely P., *Retrospective and Challenges for Model-Based Interface Development*, "Proceedings of the 3rd International Eurographics Workshop on Design, Specification and Verification of Interactive Systems", Namur, 1996, pp. 1-27.

Van den Bergh J., Luyten K., Coninx K., *Evaluation of High-Level User Interface Description Languages for Use on Mobile and Embedded Devices*, "Proceedings of the Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages", Gallipoli, 2004.

W3C, *Extensible Markup Language (XML)*, 2004. http://www.w3.org/TR/REC-xml. Download: 2005-04-13.

W3C, *XSL Transformations (XSLT)*, Version 2.0, 2003. http://www.w3.org/TR/xslt20. Download: 2005-04-15.

Ziegert T. et al., *Practical Experiences with Device Independent Authoring Concepts*, "Proceedings of the Workshop on Advanced Visual Interfaces", Gallipoli, 2004, pp. 17-24.

Zimmermann G., Vanderheiden G., Gilman A., *Universal Remote Console Prototyping of an Emerging XML Based Alternate User Interface Access Standard*, "Proceedings of the 11th International World Wide Web Conference", Honolulu, 2002. http://www2002.org/CDROM/poster/163/. Download: 2005-04-10.

JĘZYKI OPISUJĄCE INTERFEJS UŻYTKOWNIKA SŁUŻĄCE DO GENEROWANIA TREŚCI DLA RÓŻNYCH TYPÓW URZĄDZEŃ

Streszczenie

Ewolucja jaka miała miejsce w ostatnich latach w dziedzinie systemów telefonii komorkówej, rosnąca popularność urządzen mobilnych oraz coraz większe wymagania użytkownikow odnośnie dostępu do informacji przyczyniły się do gwałtownego rozwoju metod pozwalających na dostarczanie danych niezależnie od typu urządzenia. Z lamusa dawno zapomnianych metod wyciągnięto między innymi koncepcję języków opisujących interfejs użytkownika (ang. UIDL), które nieoczekiwanie znalazły szerokie zastosowanie w generowaniu treści niezależnie od rodzaju urządzenia. Niniejsza publikacja opisuje najbardziej popularne języki typu UIDL oraz oferuje ich analizę porównawczą.

Dipl-Kffr. Bożena Jankowska is a research assistant at the Chair of Business Informatics, Banking and Finance at the European University Viadrina, Frankfurt/Oder
e-mail: euv-6204@uni-ffo.de