Native End-User Languages: A Design Framework

Basawaraj Patil*, Klaus. Maetzel, Erich. J. Neuhold
German National Research Center
for Information Technology (GMD-IPSI)
Dolivostrasse 15,D-64293, Darmstadt, Germany.
{patil, maetzel, neuhold} @darmstadt.gmd.de

Keywords: POP-I.B. *barriers to programming* POP-II.A. *novice programmer and native end-users* POP-III.C. *cognitive dimensions* POP-III.B. *new language*

Abstract

In the evolving information-rich society, there are compelling reasons (e.g., educational, social, cultural etc.) to support global, native end-users to learn programming and computing skills. Current computer languages and computing environments pose conceptual, logical, semantic and syntactic challenges to native end-users, especially those who lack English knowledge and are new to programming. This paper identifies the native end-users' usability issues of programming languages and proposes a novel nativeization paradigm (NP), in which, the native end-users can command, operate, interact and program in their natural, information structures. This research describes a design framework and rationale of a simple, experimental, computer programming language based on nativeization. This paradigm will be embodied in EQUAL, a system for design, implementation and evaluation of native end-user programming languages and computing environment in a few representative native end-user languages such as German, Cyrillic, Italian and CJK (Chinese, Japanese and Korean).

Introduction

In the evolving information-rich society, computing and programming skills, i.e. computer literacy are becoming important in educational, professional and personal domains. In particular, end-users those who lack English knowledge and are new to programming have difficulties in learning computing and programming skills. Due to rapid globalisation and Internet explosion, the majority of Internet users do not use English, native end-user issues may become serious because the vast majority of the world's population who do not, and will not in the foreseeable future speak English will be excluded from the system (Dyson, 1997). People lacking computer skills may become a serious individual and social problem (Madon, 2000) and in the worst case, may lead to - Internet Apartheid (Shneiderman, 2000). In new technology adaptation, barriers from external sources may be mainly concerned with availability and accessibility of software and hardware resources (Rogers, 2000). Lack of native end-user computer languages and computing environments (e.g., interpreters, compilers, translators, etc.), native end-user interfaces and interactions, linguistic barriers, complex social and cultural factors will continue to hinder the effective learning of programming and computing skills. We also believe that user interfaces to systems for programming and computing environment can critically influence the development of programming skills. Hence, a wide range of requirements of native end-user languages, tools and computing environments must be included in the universal design and usability of native end-users information structures.

In this paper, we mainly focus on the universal design and usability issues related to programming languages, tools and computing environments from native end-users' perspectives with more emphasis on semantics and syntax of languages – i.e. notational aspects of programming and information structures. Furthermore, it will contribute to the better understanding of native end-user requirements, design of end-user languages and development tools, and implications to the scaffolding and training tools.

* Corresponding author

Native End-Users: A Profile

Research studies on native end-user requirements related to programming languages and computing environments are scarce and their requirements are poorly understood. We describe broad perspectives on native end-user, programming language issues and initiate a few basic steps in that direction. Within the scope of our research work related to universal design, universal accessibility and universal usability, we define and focus our studies to large sections of global end-user populations called *"native end-users"*. Native end-users may be defined as end-users with one or more of the following characteristics: (a) They use their first language for all human-computer interactions. (b) They have minimal or no English knowledge. (c) Their medium of education and instruction is in their native languages and socio-cultural environments. (d). They may or may not posses computing skills. (e) They would like to acquire not only information manipulation skills but also skills of computing and programming in their natural, end-user centred, textual information structures. Native end-users referred to here include students (e.g., high schools, collages and universities etc.) and adults with general educational background although they may be complete novices with respect to programming. During the in-house discussion and preliminary survey with native end-users, the majority of native end-users posed a common question " *Can we command, operate, interact and program in our linguistic, cultural and use-centred information structures"?* In response to this question, our initial answer was "yes". We believe that, by providing human-centric languages, computing environments and special tools by taking into the considerations of universal usability issues of programming languages, tools and computing environments can minimise or reduce programming difficulties of native end-users.

Nativeization Paradigm: An Effective Strategy

In order to support universal design, universal accessibility and usability of computer languages, tools and environments, we propose the concept of *"nativeization"* as an effective strategy to support native end-users. In general, nativeization may be defined as a human-centric design process of personalization and adaptation, in which the native end-users may command, operate, interact and program in their natural, information structures at conceptual, logical, semantic and syntactic levels of human-computer interactions and programming. The basic principles of nativeization paradigm are based on usability engineering, cognitive sciences, psychology of programming and general principles of HCI and human factors. This concept may be used in the design of end-user interfaces and different styles of human-computer interactions. We also exploit this concept, mainly in the design of textual languages such as command languages (Patil et al., 2001), scripting and programming languages. It should be emphasised that, by supporting only the syntactic elements of computer languages in different native end-users information constructs may be of little help, unless we support the semantics of program constructs that match with the native end-user information structures. This is particularly true for complex control and loop constructs of programming languages.

Native End- User Languages and Computing Environments

Because of historical reasons, the majority of computer languages, interpreters, compilers and integrated computing environments are based on English and English based paradigm of interactions. Current computer languages and computing environments, in particular, notational aspects of the programming language such as programming statements, variables and data values, mathematical, logical and relational operators are not available in native languages. Hence, they *"do not speak the language"* of the native end-users and do not support the basic principles of usability engineering. Thus, native end-users encounter additional difficulties and are a major issue at conceptual, cognitive, logical, semantic and syntactic levels of native end-user interactions. However programming is a very difficult activity, especially for beginners. Some of this difficulty is intrinsic to programming, but some part of it can be relieved by careful attention to accessibility and usability issues during the design of programming languages, tools and computing environments. The majority of usability problems in native end-user interactions and programming arise under the influence of language, cross-cultural, psychological and cognitive factors. But, we focus mainly on language issues for the acquisition of the means – notations of programming languages.

In general, programming is the task of mapping the mental plans and program compositions into language constructs - semantics and syntax of a computer language to achieve a particular task or

action (Ebrahimi, 1994). The characteristics of the language in which native end-users describe their plan are of critical importance. Notational aspects play a vital role in acquiring the skills of programming. Cognitive processes and structures, program constructs play a vital role in problem solving, programming and comprehension (Coombs et al., 1982; Gilmore et al., 1984). Such an understanding would lead to better programming languages whose syntactic structure more closely reflects internal semantic structures, thereby easing the process of programming.

Visual languages for programming may help to learn manipulation of information structures. But, it is observed (Myers, 1998) that the textual languages are more useful in learning the skills of programming such as comprehension, creation, documentation, modification and debugging and learning programming. It is observed that natural constructs and commands are effective in the design of end-user interactions and systems (Bierman et al., 1983; Ledgard et al., 1980; Miller, 1981; Landauer et al., 1983). Even the recent studies (Myers, 1998) demonstrate the concept of "*natural programming"* and it is useful to end-user programming and literate programming (Nardi, 1993). In teaching programming, the emphasis stressed on the means of expression can lead to an overestimation of the learning of programming language syntax within the triad determining human-computer interactions: external task structure, language syntax, and language semantics (Moran, 1981). Before venturing into a full fledged, end-user programming language, we confine our preliminary studies to a simple, experimental BASIC-like language in a few native end-user languages. We are aware of known limitations and misconceptions (Bayman et al., 1983; Ebrahimi, 1994). We have selected BASIC-like features for the experimental work because of availability of extensive psychological and empirical results (Mayer 1979 & 1981; Dyck & Mayer, 1985) on BASIC language. In the empirical studies (Dyck & Mayer, 1985), clearly shows that the microstructure of each statement (the number of actions required) and the macrostructure (the number of other statements in the program) are strongly related to the response time performance for both BASIC language and English. Apart from that, the BASIC language forms a sound basis for visual languages such as Visual Basic and is also used as an extension language in Microsoft's Word. Hence, from the above arguments, we presume that by providing natural information structures and program constructs may be of immense help to native end-users. In the following sections we identify specific program construct issues and propose some solutions.

Commands and Imperative Statements

Command languages are simple textual languages that are used extensively in operating systems and command based applications (Moran, 1981). As a central feature of many forms of communicative dialogue, names and naming have been extensively examined in a number of core disciplines such as linguistics, philosophy and empirical psychology (Black et al., 1982; Carroll, 1978 & 1982). Both formal and informal observational analyses repeatedly identified names, naming and structural contexts as an important practical problem (Rosenberg, 1982).

Majority of assignment statements (e.g., **LET, ASSIGN** etc.), input/output statements (e.g., **READ, INPUT, WRITE, PRINT, STOP, END** etc.) are simple imperative statements or commands often take the form of **VERB** <*objects*> (e.g., European languages) or <*objects*> **VERB** (e.g., Asian and Indian languages etc.) or mixed formats in command-driven applications (Patil et el., 2001). Hence, we extend these concepts and support reserve words, commands and simple assignment statements and input/output statements in end-user languages as shown in Table (I).

English	German	Italian	CJK
LET	GELASSEN	LASCIATO	Ê ¹
ASSIGN	WEISEN SIE ZU	ASSEGNARE	¸ Ö µ
READ	GELESEN	COLTO	¶ Å ¡
INPUT	INPUT	INPUT	Ê à ë
WRITE	SCHREIBEN	SCRIVONO	Ð È ë
PRINT	DRUCKEN	STAMPANO	´ Ó ¡
STOP	STOPPEN	ARRESTANO	Í Ð ¹
END	BEENDEN	CONCLUDONO FINE	½ ê ø

Table I –A set of imperative statements in native end-user languages.

Logical and Relational Operator

The accurate specification of Boolean expressions, relational operators and complex mathematical notations is a notorious problem area in programming and query languages (Pane & Myers, 2000). Researchers have observed that the common uses of the words **AND, OR**, and **NOT** and relational operators in natural language lead to errors in the use of these words to name Boolean operators in mathematical expressions and query operations. In terms of acquisition of notations, logical, relational and mathematical precursors have been shown to play an important role. The ability to use logical connectors (e.g., **AND, OR** and **NOT**) and relational operators (e.g., **LESS THEN, GREATER THEN, LESS THEN EQUAL TO, EQUAL TO** etc.) are crucial for programming notations and query operations and the majority of these problems appear to result from end user's lack of an operational understanding of DeMorgan's law of logical identities. Normally these operators are available in English and native end-users encounter additional difficulties. Hence, we believe by providing native notations of logical and relational operators may help native end-users in constructing correct logical or relational operators as shown in Table (II).

English	German	Italian	CJK
AND	UND	E	Ó ë
OR	ODER	O	» ò
NOT	NICHT	NON	· Ç
EQUAL TO	GLEICH	UGUALE A	µ Ð Ú
GREATER THEN	GROSSER ALS	PIÙ RANDE ALLORA	´ ó Ú
LESS THEN	WENIGER ALS	DI MENO ALLORA	Ð Ó Ú

Table II – A set of logical and relational operators in native end-user languages.

Control and Loop Statements

Control strategies and looping constructs very difficult concepts and demand high cognitive loads (Soloway et al., 1983). It may be observed that, just by providing equivalent control and looping constructs (**IF** <*condition*> **THEN** <*statement(s)*> **ELSE** <*statement(s)*>, **REPEAT** <*statement(s)*> **UNTIL** <*condition*>, **WHILE** <*condition*> **DO** <*statement(s)*> etc.) in native end-user information structures may be insufficient and inadequate, some time quite misleading and confusing from the preconceived knowledge (Bonar et al., 1985). In the studies (Wu, 1991; Rogalski, 1990), clearly demonstrates different loop exit possibilities (e.g., top-exit, bottom-exit or mixed exit) and it is observed that, top-exit strategy is more difficult then the bottom-exit strategy. In **IF-THEN** statement or its variants, it is observed that (Boulay, 1989), end-users use **THEN** for "sequencing" and contradicts the meaning of "consequently" in formal constructs. Careful design and extensive research studies are necessary to explore the implications of these information structures and program constructs in native end-user languages. In our future work, we explore the relationship between semantics of loop and control structures in native end-user languages and formal semantics and develop an effective mapping into the formal semantics of computer languages. In the present state of research, explicit training is the best way to circumvent these difficulties.

Native-End User Program: An Example

From the above discussions, we represent experimental, program constructs in few representative native end-users' languages. Native end-users can write their programs using a Unicode compatible text editor or a special editor and store the program in a Unicode format. The native end-user files may further processed by the compiler or interpreter for syntactic or semantic analysis of programs. The errors, warnings, results of executions, on-line documentation may be available in native end-users languages. Since native end-users are able to write (e.g., names of variables, data values, numerical values etc.) in native end-user languages in a more meaningful and are useful in program comprehension (Gellenbeck & Cook, 1991). Important salient features of EQAUL language and program constructs are shown in Table (III).

English	German	Italian	CJK
LET length = 20; LET breadth = 45; Area = length * breadth; PRINT length, breadth, area; PRINTALL END	LASSEN SIE Länge =20; LASSEN SIE Breite = 45; Bereich = Länge * Breite; DRUCK Länge, Breite, Bereich; DRUCK ALLE ENDE	Lunghezza = 20; Lasciare la larghezza = 45; Zona = lunghezza * larghezza; Lunghezza della STAMPA, larghezza, zona; FINE	¸ Ö µ¨ ¶ È20; ¸ Ö µ¿ ¶ È45; Ã æý³ ¶ È¿ í ¶ È ´ Ö ¡³ ¶ È¿ í ¶ ÈÃ æý È « ¿ Ö ¡ ½ È ø

Table III –EQUAL computer programs in native end- user languages.

EQUAL Language Design Principles

After discussing important native end-user issues and providing some solutions, we discuss the following native end-user language design guidelines and basic principles. It should be again emphasised that nativeization and its exploitation are one of the main guiding design principles of EQUAL language.

Develop Simple Language: Keep syntactic elements and semantic concepts as simple as possible. Avoid complex data structures and data types, complex and nested loop and control structures and recursions.

Support High Adaptability: The languages, tools and environments shall be error tolerant, support high degree of personalization, adoption and flexibility.

Speak Native End-Users' Languages: It is a well-known design guideline from usability engineering and plays a vital role in native end-user languages and computing environments. The native end-user must be able to command, operate, interact and program in their native end-users' information structures. Support also end-user's task characteristics.

Exploit Cognitive Dimension of Notations: The process of translating a mental plan into one that is compatible with the computer is a highly cognitive activity. The language should minimise the difficulty of this translation by providing high-level primitives that match the operators in the plan and composition. Especially take in to considerations of different attributes of cognitive notations (Green, 2000).

Support Native End-User Environments: Just providing an end-user computer language is not enough, a corresponding computing environment such as interpreters, compilers and integrated environment are required. The error messages, warnings, results of the execution, and on-line documentation with examples must be available in the native end-user languages. The development environment of native end-user computer languages shall be flexible enough to accommodate diverse or new requirements of native end-users.

Separate Native End-User Dependent Information or Data: Using the principle of separation of concern, separate all the native end-user dependent information and data (e.g., linguistic, morphological, cultural etc.) into separate software modules and store them in an external database. The database may be enriched and extended by the domain-experts (e.g., linguistics, cognitive scientists, HCI and Human Factor experts etc.)

Discussion and Future Work

In this paper, we discussed the universal usability issues of programming languages and computing environments from native end-user's perspectives. We have developed a design framework and rationale of native end-users' languages and computing environments based on the concept of

nativeization and also highlighted some of the the implementation issues. We also discussed the salient features of EQUAL language for native end-users. In future, we explore the relationship between loop and control structures in native end-users languages and formal programming constructs. We also study and develop effective solutions to map the loop and control native information structures onto the formal semantics.

From the implementation point of view, the proposed framework may be implemented as a translator system that maps the native end-users information structures to the back-end computing languages, tools and environments. Another effective approach is to develop an abstract representation of programming constructs and information structures and derive a concrete realisation by mapping native end-users information structures using latest Java, Unicode and compiler technologies. Experimental work is in progress and we have implemented a few native end-user information structures (e.g., commands and imperative statements, naming of variables and data values, relational and Boolean operators etc.) in a few end-user languages. Implementation of complex data types, control constructs and critical evaluation are planned in the future work.

Acknowledgements

This work is supported by the GMD - German National Research Centre for Information Technology, Darmstadt, Germany. I am grateful to my advisors Prof. Erich. J. Neuhold and Prof. Dr.Klaus Maetzel for their motivation and support of my work. I thank many foreign colleagues and friends for their valuable linguistic translation help and participating in brain storming sessions and survey.

References

Bayman, P. & Mayer, R.E. (1983), A Diagnosis of Beginning Programmers' Misconceptions of BASIC Programming Statements, *Communications of the ACM*, Vol. 26(9),pp. 677-679.

Biermann, A.W., Ballard, W.B.W., & Sigmon, A.H. (1983), An Experimental Study of Natural Language Programming, *International Journal of Man-Machine Studies*, 18, pp.71-87.

Black, J. & Moran, T. (1982), "Learning and Remembering Command Names" in *Proceedings of Human Factors in Computer Science* (Gaithersburg), ACM, New York, pp.8 – 11.

Bonar, J.& Soloway, E. (1985), Pre-Programming Knowledge: A Major Source of Misconceptions in Novice Programmers, *Human – Computer Interaction*, Vol. 1,pp. 133-161.

Boulay, B. (1989), Some Difficulties of Learning to Program, In Soloway, E. & Sphrer, J.C (Eds.), *Studying the Novice Programmer*, Lawrence Erlbaum Associates, Hillsdale, New Jersey,pp.283-299.

Carroll, J.M. (1978), Names and Naming*: An Interdisciplinary Review.* IBM Research Report RC 7370, 1978.

Carroll, J.M. (1982a), Learning, Using and Designing Command Paradigms, *Human Learning*, 1, pp.31-62.

Coombs, M.J., Gibson,R. & Alty, J.L. (1982), Learning a First Computer Language: Strategies for Making Sense, *International Journal of Man-Machine Studies*, Vol. 16, pp.449-486.

Dyck, J.L, Mayer, R.E. (1985), BASIC Versus Natural Language: Is There One Underlying Comprehension Process, *Human Factors in Computing Systems – II, Proceedings of the CHI'85*, San Francisco, USA, pp.221-223.

Dyson, E. (1997), Education and Jobs in the Digitial World, *Communications of the ACM*, Vol. 40(2), pp.35-36.

Ebrahimi, A. (1994), Novoice Programmer Errors: Language Constructs and Plan Composition, *International Journal of Human-Computer Studies*, 41, pp.457-480.

Gellenbeck, E.M. & Cook, C.R. (1991), An Investigation of Procedure and Variable Names as Beacons during Program Comphrension, *Empirical Studies of Programming: Fourth Workshop*, Koenemann, J.B, Moher, T.G. & Robertson, S.P. (Eds.),Ablex Publishing Corporation, New Brunwick, New Jersey, pp.65-81.

Gilmore, D. J. & Green, T.R.G. (1984), The Comprehensibility of Programming Notations, *HCI-Interact'84: Proceedings of the IFIP Conference Organised by Task Group on HCI*, London, U.K, pp.461-464.

Green, T.R.G (2000), A *Cognitive Dimensions Web Site*. http://www.ndirect.co.uk/~thomas.green/workStuff/res-CDs.html

Landauer, T.K., Glotti, K.M. & Hartwell, S. (1983), Natural Command Names and Initial Learning: A Study of Text Editing Terms. *Communications of the ACM*, 26, pp.495-503.

Ledgard, H., Whitehead, J.A., Singer, A. & Seymour,. W. (1980), The Natural Language of Interactive Systems, *Communications of the ACM,* Vol 23,pp.556-563.

Madon, S. (2000), The Internet and Socio-Economic Development: Exploring the Interaction, *Information Technology & People,* Vol.13(2), pp.85-101.

Mayer, R.E. (1981), How Novices Learn Computer Programming, *Computing Surveys,* Vol. 13(1), pp.121-141.

Mayer, R.E.(1979), A Psychology of Learning BASIC, *Communications of the ACM,* Vol. 22(11), pp.589-593.

Miller, L.A.(1981) Natural Language Programming: Styles, Strategies and Contrasts, *IBM Journal,* Vol. 20(2),pp.184-215.

Moran, T.P. (1981), The Command Langauge Grammar: A Representation of the User Interface of Computer Systems, *International Journal of Man-Machine Studies,* 15,pp.3-50.

Myers, B.A. (1998), Natural Programming http://www.cs.cmu.edu/~NatProg

Nardi, B.A. (1993), A Small Matter of Programming, The MIT Press, Cambridge, England.

Pane, J.F & Myers, B.A. (2000), Improving User Performance on Boolean Queries, *CHI'2000 Extended Abstracts: Conference on Human Factors in Computing Systems*, Szwillus,G. and Turner, T. (Eds), The Hague, Netherlands, ACM Press, pp.269-270.

Patil, Basawaraj., Maetzel, K. & Neuhold, E.J. (2001), Design and Implemenation of Universal End-User Commands, Interfaces and Interactions, *Submitted for UAHCI, 2001, New Orleans, USA.*

Rogalski, J. & Samurcay, R. (1994), Acquisition of programming Knowledge and Skills. *Psychology of Programming,* Hoc, J.M.,green, T.R.G., Samurcay, R. & Gilmore, D.J (Eds), Academic Press, London, pp.157-174.

Rogers, P.L. (2000), Barriers to Adopting Emerging Technologies in Education. *Journal of Educational Computing Research*, Vol.22(4), pp.455-472.

Rosenberg. J. (1982), Evaluating the Suggestiveness of Command Names, *Behavior and Information Technology,* 1, pp.370-400.

Shneiderman, B. (2000), Universal Usability, *Communications of ACM*, Vol. 43(5),pp.85-91.

Soloway, E., Bonar, J., & Ehrlich,K. (1983), Cognative Strategies and Looping Constructs:An Emperical Study, *Communications of the ACM,* Vol. 26(11), pp.853-860.

Wu, Q. & Anderson, J.R. (1991),Strategy Selection and Change in Pacal Programming, *Empirical Studies of Programming: Fourth Workshop*, Koenemann, J.B, Moher, T.G. & Robertson, S.P. (Eds.), Ablex Publishing Corporation, New Brunwick, New Jersey, pp.227-238.