About Ergonomics of Computer Interfaces Designing a System: Designing a System for Human Computational Collective Use

Christophe Jouis [†]
Presidency
Sorbonne Nouvelle University &
CAMS (CNRS/EHESS/Sorbonne
Universités),
Paris, France
cjouis45@gmail.com

Mercedes Orus-Lacord
Mathematics
Universitat Oberta de Catalunya,
Universidad Nacional de
Educacióna Distancia
Barcelona, Spain
mercedes.orus@gmail.com

Steven Pemberton
Computer Science/Department of
Algorithmics and Architecture
Centrum Wiskunde & Informatica
(CWI), Amsterdam, Netherlands
steven.pemberton@cwi.nl

ABSTRACT

This document discusses the ergonomic problems with currently available software products, and what in general is necessary to make an application pleasant to work with. The application of these principles to a new open-architecture user interface system, Views, is then described.

CCS CONCEPTS

Information systems • World Wide Web • Web mining

KEYWORDS

Ergonomics Computer interfaces, human interaction, Human machine interactions, collective human interactions

ACM Reference format:
Christophe Jouis, Mercedes Orus-Lacord and Steven Pemberton. 2022. About Ergonomics of Computer Interfaces Designing a System: Designing a System for Human Computational Collective Use. In Proceedings of the 14th International Conference on Management of Digital EcoSystems (MEDES'22). ACM, Venice, Italy.

1 Introduction

Current software products and applications are typically 'non-cooperating', in the sense that each has its own usage conventions and data-formats, even in computer environments (such as Mac OS) where interface standards are clearly defined.

MEDES '22, October 19–21, 2022, Venice, Italy
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9219-8/22/10…$15.00
https://doi.org/10.1145/3508397.3564823

This lack of uniformity puts a high cognitive load on the user, increasing errors, reducing productivity, and limiting the potential users of a system.

This document discusses what makes a system pleasant to use in general, largely abstracting from the specifics of any application context and presents the ideas behind a new open-architecture user interface system, views. Ideally, the views system will bring the users in control of their environment and offer them a completely unified user interface for all applications running under it.

2 The changing environment

Within this century workstations that by today' s standards are extremely powerful welcome to occupy a place in virtually every employee's working environment. This massive 'invasion' will be caused, of course, by the continuing rapid drop in prices.

With current prices, equipping every of office worker with their own powerful workstation would be economically entirely unfeasible, whereas providing them all with now affordable microcomputers would not result in productive use of the new technology. The time will inevitably come when a fordable systems become available that can increase the productivity of almost every employee. Once this point has been reached, it is to be expected that the 'take over" will take place with unprecedented speed, even amongst. those who have sworn never to touch a computer. The key issue here is the productive use of the relatively new computer technology. Currently, computers are used productively by direct users (in contrast to being embedded in a larger system) only if at least one of the following four criteria is met:

- the user happens to be an able programmer,
- the task involves heavy computation, usually of a scientific nature,
- the task is of an entirely routine character, usually but not necessarily carried outby several 'interchangeable' workers collaterally,
- the task is one of a small list of ubiquitous 'standard applications', like word-processing and spreadsheets.

In due time software products will appear with which many potential computer-users can make productive use of a fordable systems. This is clearly inevitable, since intend years from now there will be a mass demand for such products. The availability of such products is, conversely, instrumental to the above-mentioned 'take over', and so they will help to create their own market, in the same way that spread sheets and word-processing software have already advanced the sales of personal computers. With that 'take over', a change in the software market that already has started to take place will become irreversible.

The software market will then become and stay predominantly amass market, on which the quality of products will be assessed more by their ability to meet the manifold needs of an individual user than by the extent to which they conform to specific needs of organizational entities.

Although there is no reason to assume that the turn-over of custom-made applications will decrease in absolute terms, its relative significance on the overall market will diminish. Yet another change in the shape of the market is predictable. Whereas noncommissioned software is mainly produced by the larger software houses, and many if not most mass soft-ware products are — at least initially — produced by very small groups, or some-times even a single person, these roles will tend to be reversed. The kind of software ergonomic principle that will meet the requirements of the indicated future market segment is so complex, and them competition will be so intense, that only the stronger, better prepared producers will be able to acquire a share of the newly opened market.

3. Objectives of Views

The Views project [1] aims at creating a framework for the user interface of applications such that the system (given sufficiently powerful microcomputers or workstations) can be put to productive use by end users without much training in using computers, for varying tasks that are relatively complicated and not well structured.

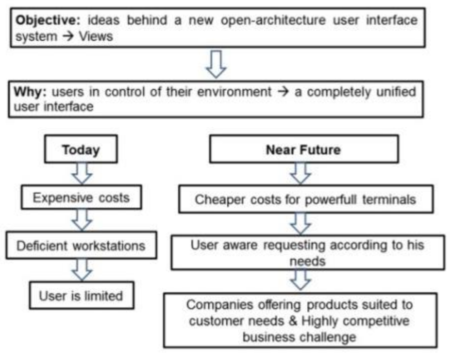
This is to be achieved by reaching a large and hitherto unparalleled degree of ease for the end user in controlling the environment, through the consistent uniformity and extreme conceptual simplicity of the interface and the high level of integration between functions of the environment. The objective of views, then, is the development of a conceptual framework and a methodology for designing and implementing integrated applications in the form of a 'framework' user-controlled operating environment that allows easy addition of functions and tools (including end-user applications) in such a way that the desired cooperation and uniformity is guaranteed, and the design of a single uniform interface both for built-in functions and for functions added later.

The architectural framework of views can be described as an open system architecture, with interface standards both for intra-system data exchange and for user-system communication, in which it is very easy to add applications conforming to these standards.

This ensures that it is possible to maintain a very high level of integration be-tween various applications, including standard services.

Views is equipped with several standard services and document types, making it flexible and valuable operating environment even if no special-purpose applications are added.

The following figure summarizes the objective of this article.


Objective: ideas behind a new open-architecture user interface system → Views
Why: users in control of their environment → a completely unified user interface
Today
Expensive costs
Deficient workstations
User is limited
Near Future
Cheaper costs for powerfull terminals
User aware requesting according to his needs
Companies offering products suited to customer needs & Highly competitive business challenge
Figure 1: Objectives and Why

4. Ergonomic principles

Although software ergonomics as a science is still in its infancy, the active research in this area has already produced many valuable insights concerning factors that in-fluence the productivity in the use of software. In particular, the research has concentrated on the cognitive ergonomic aspects of the user interface (or human-computer interface). One of the lessons that has been learned is that human beings are quite varied (for example, in planning: ad-hoc vs. planning ahead; and in cognitive abilities or skills, like the skill to use formal systems or the ability to use spatial cues to their advantage) [2].

Most competent, intelligent, and successful people score low on one or more of these many skills. In coping with life, they often have bypass strategies: they use other skills to overcome deficiencies. Many user interfaces are unsatisfactory be-because they are too narrow: they require a certain fixed combination of skills and al-low no substitutes. A good interface must cater for a variety of users. In the use of computers for solving tasks we can distinguish between more human-oriented and more computer-oriented approaches in human-computer interfaces. A human-oriented task analysis is a key issue in the design of good user interfaces. A substantial part of this analysis can be made relatively domain-independent, and thus it is possible to obtain insights of a general validity that have a bearing on the de-sign of user interfaces, such as: evaluation is an integral part of human cognitive behaviors; humans are in general constitutionally unable to perform a complex task according to a fixed 'program' (stack-wise, or breadth-first); switching task con-texts in the middle of a task is expensive and error-prone.

From such ergonomic principles, it is not hard to understand why it is that current software cannot be put to sufficiently productive use in a context of varied and ill-structured tasks. Looking at typical present-day products, we find the following kinds of shortcomings.

To obtain the desired results, end users often must combine the workings of sever-al applications. Each application has its own requirements on the format of the da-ta. This often requires a manual intervention of the user, for example by means of a text editor, to massage the data output by one application into some required input format for another.

Another problem is that once the data has been imported, its structure is often lost therefore, so that to make a change the data must be reimported, rather than changing it in situ.

Each application and the operating system itself have their own mode of addressing, and some applications have several such modes, exacting from the user not only knowledge about all these modes and the way of expressing commands and functions under these modes, but also always awareness of the current application and mode, which is ahigh cognitive load, and thus a source of errors [3].

It is the rule rather than the exception that in performing a task a user finds some-where on the way that certain data needed to proceed is not immediately available and must be obtained somehow outside the context of the current task. Many systems allow such task switching only if the pending task is first brought into some well-defined, quiescent state: rather than being suspended, it must be ‘closed’. This means then that the context of the unfinished task is lost and must be explicitly re-stored by the user at the time it is resumed.

The phenomenon of inflexible software, especially as regards the input format, but also concerning the order in which the user can take certain steps, is so well known that it need not be labored here. The exact forms or formats required for commands or functions that are not regularly used are often such that it is impossible to re-member them when they are needed, so that they must be dug up from huge and usually hard-to-interpret manuals.

Many software applications ‘package’ functions, like, for email, finding a letter (retrieval), archiving it (storing), composing a letter (editing), etc. Such functions are in fact emasculated versions of much more general functions that happen to be ap-plied to one specific context. This can lead to a tremendous duplication of functionality, and increased inconsistency, entailing the constant danger of mode confusion and adding to the overall complexity of the user interface.

It is almost as if these shortcomings have been purposely designed with one single objective in mind: to thwart users in their attempts to make productive use of technology. For together they conspire to put such a cognitive load on the user, distracting from the real task to be performed, that a large rate of user mistakes becomes unavoidable. It is especially at routine tasks that computers easily outperform human beings, who start making errors due to waning attention just in repetitive tasks, and yet a substantial part of the effort in using computers as an intellectual tool consists of clerical tasks without intellectual content, like transforming data formats. The current situation forces users to spend a substantial part of their attention continually on low-level aspects of the communication, even down to the motoric level, rather than on the conceptual level of the task to be performed. While computers are far better than humans in keeping track of the precise status of a task in operation, it is the user who is forced to commit the details of the context to memory, not the computer. The mistakes made are most of the time just ‘silly’ mistakes, causing annoyance and loss of time but no great damage. Every now and then, however, an entirely understandable user mistake is costly, causing wrong results, or the loss of much work, or even irretrievable loss of information, causing great anguish. Even without such mishaps, the total time users spend on guessing, being puzzled, or trying to find out something quite trivial, is staggering. However, these ‘unavoidable’ mistakes are unavoidable only because of the failure to ap-ply the most basic ergonomic principles to the design of the user interface [4]. An important prerequisite for a good interface is that the system appears to the user as an integrated system with a single interface, rather than a collection of disparate interfaces, one for each different function of the system, however well designed each individual interface may be. This, then, requires in its turn that one single conceptual framework underlie the whole system, a framework in which diverse applications can be integrated in natural way.

The following figure summarizes some topics of this section.

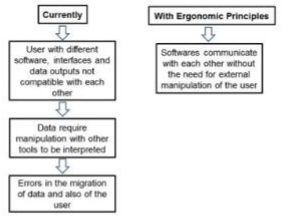

Currently
User with different software, interfaces and data outputs not compatible with each other
Data require manipulation with other tools to be interpreted
Errors in the migration of data and also of the user
With Ergonomic Principles
Softwares communicate with each other without the need for external manipulation of the user

Figure 2: Ergonomic Principles

5. New trends in user interfaces

The new trends in user interfaces are the result of the attempts of software designers to overcome some of the problems mentioned above. The term ‘WYSIWYG’, What You See Is What You Get, is usually applied to text formatting systems in which a document being edited appears on the screen as it will appear when printed in hardcopy form. The popularity of WYSIWYG systems is due to the immediate feedback given to the user. Although it may be argued that the amount of user control in more traditional systems is equally large, this argument denies the psychological reality of the immediacy of control. Since silly mistakes are directly apparent, they can be corrected instantly in the WYSIWYG approach, whereas the users of the traditional systems must re-create their intention within the context under the indirect edit-print-proofread-correct loop. Also, the possibility of repeating a particular mistake consistently throughout the preparation of a document becomes remote. The same principle can be applied in a much broader sense to all kinds of systems in which the effect, in logical terms, of a user action is made instantly visible, providing feedback. In that sense all arcade-type of computer games are examples of WYSIWYG systems, but also spreadsheet applications that continually update the derived information. A further extension of the principle could be stated as ‘TAXATA’: Things Are exactly As They Appear. It may seem that this is a consequence of WYSIWYG, but in practice the equivalence is effective in one direction only: the effect of a user action is made immediately visible to that user, but not the effect of changes from other causes. In fact, it is usually the case that the user edits a copy of the document, rather than the document itself, so that such changes could not become visible. If the operating environment permits a task switch without leaving the edit application, the co-existence of two versions may then lead to user confusion. Rather than giving commands in some arcane command language to a slave who, if noncomplaining about syntax, scurries off to execute the command in the hidden dungeons of the computer, the user 'directly executes', for example, the printing of a document by dragging the icon of the document to the icon of a printer. Although it is still the computer that executes the command according to the directions given by the user, psychologically it is the same as the real physical execution of the action by the user. To use a metaphor, the traditional dialogue mode is like taking a taxi ride with a driver who must be told to turn left or right all the way while direct manipulation is like driving the car yourself. An important advantage of direct manipulation is that there is much less chance of making an error in phrasing the command, there is a notable increase in the uniformity of the external form of certain functions that can be modelled on the same or simi-lar metaphors, and the user gets explicit visual clues on how to express the action. The experience has been indeed that novice users can learn to use systems based on direct manipulation productively in a fraction of the time needed for more conventional systems. As with WYSIWYG, the notion of direct manipulation is still used in a narrower sense than warranted: restricted to spatial metaphors such as with icons on a desktop and shoving them around. A much more common form of user control that can also be viewed as being a form of direct manipulation is that of entering text while editing a document: each key hit appears on the screen, and to the user this is psychologically no less direct than when using a mechanical typewriter. Yet another form is that in which modifying the name of a document as it appears in a caption while it is displayed also renames the document name as it is stored in the archiving system. The common aspect here is that the psychologically physical action by the user (moving an icon, hitting a key, changing a name) has a direct effect in the universe of the computer, which obeys certain logical, as it were pseudo-physical, laws. Indeed, in the real world physically changing the name of a document, say the title of a book on a shelf, causes that book to carry a different title. This last example can also be seen as an application of the WYSIWYG principle: what you see as title in the caption is what you getas title for the document. Carried to their logical extremes, it can be argued that the WYSIWYG and direct manipulation coincide.

The term 'integrated' is increasingly used as a sales argument for systems that do not merit this epithet. Yet it is not a buzzword, but has a clear and definite meaning, albeit hard to formalize. An information system is integrated if it presents itself to the end user as a collection of functions and tools that can cooperate smoothly and that can be handled by the user in a uniform way. It is a well-known and not surprising fact that software packages usually do not cooperate smoothly. Each time, a separate substantial effort is needed to get them to cooperate. This sometimes requires access to the source code, which may be unavailable, and generally requires skills that can-not be expected from end users. In view of the consequences of lack of integration for user productivity, it is easy to understand that truly integrated systems are among the best-selling software products. Available integrated systems consist of some cooperating functions and tools (e.g., a text editor, a spreadsheet part, and a database part), but are not extensible with new functions and tools. Typically, each package has a 'friendly' user interface, but then each package has its own conventions, or a rigid interface that cannot easily be adopted to another style. Although modern interfaces have a WYSIWYG approach, this is usually only partially pursued and not fully taken advantage of.

In contrast to the preceding terms mentioned, the properties that warrant calling a system 'intelligent' are not well delineated, and it may be argued that even the least dumb systems are not worthy of being ascribed intelligence. But, at least, a trend is discernible from exceedingly dumb towards somewhat less exceedingly dumb, and therefore more intelligent, systems. To make this more precise, the first step is to make clear that the 'intelligence' is to be understood in a very narrow sense, namely that of helping the user in performing a specific intellectual task, consisting of the (usually repeated) identification, evaluation, and selection of alternatives, usually with the additional requirement that nondecisions are definitive but can be changed or un-done at any time. There are many examples of intellectual tasks that fit this description: all kinds of design (not only industrial, but also system design, or the design of, say, a marketing tactic, a departmental reorganization, or the floor plan of a stand on an exhibition), all kinds of planning and scheduling (work charts, rosters, assignments, routing), packing problems, and tasks involving combinations of such problems. (It would not be unreasonable to call systems for providing support to such tasks generically 'Decision Support Systems', but currently DSS's are generally understood to address specifically management decision tasks and include kinds of systems that do not fit the above characterization.). Still, few, such 'intelligent' systems will become increasingly important in the whole software market. The diversity is too large to handle this category in a uniform way, especially since successful programs in this category concentrate on a single type of task, but the common aspect and the key to their success is that these programs combine domain-specific expertise (often not only in the form of clever algorithms or an embedded expert system, but also in the form of brute force, made feasible by the dramatic increase in computing power) with a highly visual and interactive user interface. The user has the freedom to try certain decisions and is presented (ideally instantaneously) with the effects of that decision in combination with the earlier decisions taken and with an overview of the remaining design freedom given the constraints of the problem. This corresponds to what is known in the DSS world as a 'What If' facility. Thus, this form of user-system interaction, which is crucial to the productive use of such systems, is a com-bination of the generalization described above of the WYSIWYG and Direct Manipulation principles, where the laws referred to are not given once and for all but are specific to a given problem setting. It is also worth remarking that for simple problems of this general kind a spreadsheet approach may be quite satisfactory. Although there is a reasonably common understanding what is meant by 'Fourth-Generation Languages' (or 'Systems') [5], there is also a surprising agreement that most systems marketed as such fall short of their promise. Still, whether such systems meet the agreed definition or not, they often succeed in what is their primary aim: to affect a dramatic increase in programmer productivity. They have a limited domain of ap-applicability, which, however, is important by virtue of the number of applications fitting that domain. Fourth-Generation Languages/Systems provide a convenient way to describe certain applications (which, if that is what is emphasized, gives rise to calling it a 'Language'), and these descriptions can be used to generate programs realizing the application (whence 'Systems'). The success is also due to a form of integration, albeit of a different form than described above. Here, the integration is achieved, not by the uniformity of data exchange between the parts of the final application (such as a data-entry part, a database, and a report generator), but by the fact that the pro-gram generator also generates the programs for the necessary adaptations of data for-mats between these parts, thus taking away this drudgery from the application programmer.

The following figure summarizes some topics of this section.

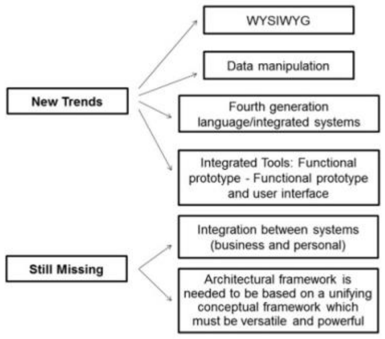

WYSIWYG
Data manipulation
New Trends
Fourth generation language/integrated systems
Integrated Tools: Functional prototype - Functional prototype and user interface
Integration between systems (business and personal)
Still Missing
Architectural framework is needed to be based on a unifying conceptual framework which must be versatile and powerful

Figure 3: New trends

The kind of application generated is usually highly conventional and not substantially different from applications as they were written fifteen years ago; what has changed is the time needed to produce them. The obvious next step is to use similar techniques in a more flexible way to generate more diverse and modern applications, and a few software products that have taken large strides in this respect have already appeared. A further obvious step that will inevitably be taken sooner or later is the integration of intelligent tools for system design with such sophisticated application generators.

The term 'Rapid Prototyping' is used in two related but distinguishable senses. The first is that in which a fast technique is used to realize a fully functional but possibly inefficient prototype of an information system. This makes it possible to recognize and correct certain shortcomings, for example in the user interface, before tremendous investments have been spent on the coding phase. Increasingly, it is the case that the prototype that is deemed satisfactory (from a functional point of view) after several iterations is not thrown away but is used as starting point for a production version by identifying bottlenecks and crucially time-efficient parts of the system and recoding only those. The second meaning is that in which it is possible to create a mock-up version that is not fully functional but models, for example, only the user interface. In such a case, shortcomings in the user interface can be identified in a timely fashion. This is important since experience has shown both that a large part of the major deficiencies tends to be in the user interface, and that coding the user interface is relatively costly [6][7]. The distinctions not sharp, and it is increasingly possible to 'grow' prototypes by piecemeal addition of functionality. In a sense, Rapid Prototyping of the first type is what Fourth Generation Systems try to achieve, with the addition that the latter already aim (not always quite successfully) at production-quality code for the generated result. If Rapid Prototyping is seen as distinct, this is because it usually has less strict limitations on the application domain, and because, as a result, the techniques are less complete.

The wider group of end users and the attendant larger variety in tasks has spurred research into methods by which end users can build their own relatively simple applications from available building blocks. This is basically a form of programming, and although programming is intrinsically hard, this task can be eased in several ways, to an extent that it comes within reach of many end users.

To result in a framework that can serve as the basis for a viable product, the research carried out must take all the trends sketched in the previous section into account. A single significant shortcoming in any one of these aspects may be sufficient to doom products to eventual failure on the future market. It is therefore mandatory that the architectural framework be based on a unifying conceptual framework that is so versatile and powerful and nevertheless conceptually so simple that it can be established a prior that its embodiment will allow to meet the most demanding requirements.

6. Discussion: From office automation top Personal computing

As has become apparent from the preceding points, an emergent property of the framework system to be designed is that all kinds of data are integrated in the system, and that the user interface for these is uniform. In a system built around this frame-work, containing as subsystems for example a manager's Personal Information System (agenda with reminder service, small personal data bases, memos, annotations with half-baked ideas, spreadsheets, etc.) and Office Automation System and a Management Information System, the boundaries between these subsystems are not real, and in fact they are all integrated, so that, e.g., a memo can be dropped in the secretary's mailbox, or the data fora spreadsheet calculation can be read in from a database kept in the Management Information System. Only one control interface must be learned for all, and data can be transferred without further ado. The Object-Centered paradigm provides an easy and natural way to specify the constraints of the Office Automation System and the links with the

Management Information System and the Personal Information Systems (for example, by automatically generating overviews at set times, or reminders).

ACKNOWLEDGMENTS

This work was made possible thanks to the support of:

- Centrum Wiskunde & Informatica (CWI), Amsterdam, Netherlands,

- "La Fabrique Défense", French Army, which provided funds to EC3 Project,

- The EC3 Project company.

REFERENCES

[1] Steven Pemberton, Views: An Open-Architecture User -Interface System. In proceedings “Interacting with Computers: Preparing for the Nineties”, Noordwijkerhout, December 1990.
[2] Jakob Nielsen, The Matters that Really Matter for Hypertext Usability. In Proceedings of Hypertext ‘89, Second ACM Conference on Hypertext, Pittsburgh, Pennsylvania, pages 239—248, November 1989.
[3] A.J. Sellen, G.P. Kurtenbach and W.A.S. Buxton, The Role of Visual and Kinesthetic Feedback in the Prevention of Mode Errors. In Proceedings Interact ‘90, Cam-bridge, UK, pages 667—673, August 1990.
[4] D. Norman, The Psychology of Everyday Things. Basic Books, NY, 1988.
[5] K.M. Misra and P.J. Jalics, Third Generation versus Fourth Generation Software Development. IEEE Software, July 1988.
[6] M. Flecchia and D. Bergeron, Specifying Complex Dialogs in ALGEA. In Proceedings Human Factors in Computing Systems and Graphics Interface (CHI+GI‘87), Toronto, Canada, pages 229—234, April 1987.
[7] J.A. Sutton and R.H. Sprague Jr., A study of Display Generation and Management in Interactive Business Applications. IBM Research Report RJ2392(31804), Yorktown Heights, N.Y.
[8] J.J. van Griethuysen, editor, Concepts and Terminology for the Conceptual Schema and the Information Base. ISO TC97 / SC5 / WG3 American National Standards Institute, New York, March 1982.
[9] Lambert Meertens, Steven Pemberton, Guido van Rossum, The ABC Structure Editor. CWI Report CS-R9256, CWI, Amsterdam, December 1992.
[10] L.G.L.T Meertens, S. Pemberton. An Implementation of the B Programming Language. In Proceedings USENIX Conference Washington, January 1984, Also Note CS-N8406, CWI, Amsterdam, June 1984.
[11] C. Jouis, M. Orus-Lacord, N. Durglishvili & R. Orus, Management of Big Textual Data, In Qualitative Research: Organizing the Relationships In A Typology Based On Logical Properties, 11th International Conference on Management of Digital Ecosystems (MEDES '19), November 12–14, 2019At: Limassol, Cyprus
[12] L.G.L.T. Meertens, S. Pemberton, The ergonomics of computer interfaces. De-signing a system for human use, report (1992): Computer Science/Department of Algorithmics and Architecture, CS-R9258 1992
[13] H. Fadhili, C. Jouis, 2016, Towards an automatic analyze and standardization of unstructured data in the context of big and linked data, Proceedings of the 8th ACM International Conference on Management of Digital Ecosystems, November 2016, Henday, France
[14] C. Jouis, Orús-Lacort, N. Durglishvili& R. Orus, MANAGEMENT OF BIG TEXTUAL DATA IN QUALITATIVE RESEARCH: ORGANIZING THE RELATIONSHIPS IN A TYPOLOGY BASED ON LOGICAL PROPERTIES, , Proceedings of the 11th ACM International Conference on Management of Digital Ecosystems, November 2019, Limassol, Cyprus