Graceful Degradation of User Interfaces as a Design Method for Multiplatform Systems

Murielle Florins Jean Vanderdonckt
IAG – School of Management – Université catholique de Louvain
Place des Doyens, 1
B-1348 Louvain-la-Neuve, Belgium
{florins,vanderdonckt}@isys.ucl.ac.be

ABSTRACT

This paper introduces and describes the notion of graceful degradation as a method for supporting the design of user interfaces for multiplatform systems when the capabilities of each platform are very different. The approach is based on a set of transformational rules applied to a single user interface designed for the less constraint platform. A major concern of the graceful degradation approach is to guarantee a maximal continuity between the platform specific versions of the user interface. In order to guarantee the continuity property, a priority ordering between rules is proposed. That ordering permits to apply first the rules with a minimal impact on the multiplatform system continuity.

Categories and Subject Descriptors
D.2.2 [**Design Tools and techniques**]: User Interfaces

General Terms: Design, Human Factors, Theory.

Keywords: Continuity, design, graceful degradation, multiple computing platforms, multiplatform systems.

1. INTRODUCTION

An increasing number of applications can be accessed from a wide range of platforms. A *platform* is generally defined as a specific combination of hardware and operating system. When considering user interfaces (UIs), it is useful to add to this notion of platform other elements such as the browser or the available graphical toolkit(s). Sometimes, the capabilities of each platform are very different [5]: the devices differ in screen size, resolution or color number, some HTML code is rendered in a different way depending on the interpreting browser, some widgets are available within a given toolkit and unavailable within another one.

On the other side, users expect to be able to reuse their knowledge of a given version of the system when using the same service on another platform. Thus, the transitions between system versions have to be as smooth as possible. In the literature, that property of a multiplatform system is called the *continuity* property [4].

We propose an approach called *design by graceful degradation* as method to build usable user interfaces for multiplatform systems while guaranteeing the continuity between the system versions.

The remainder of this paper is structured as follows: we first report the various approaches that can be used to develop UI for multiplatform systems, then we define the graceful degradation approach. Transformation rules for graceful degradation are then identified and their impact on the continuity property is considered.

2. PAST LESSONS FROM HCI RESEARCH AND PRACTICE

Several techniques and tools have been used to develop UI for multiple platforms, e.g. [1,2,7,8,11,12,15,17] to name a few. One approach is the development of *a specific interface for each platform*. This approach does not guarantee any consistency between the different target specific UIs.

Another approach consists of designing *a single interface that will run on several platforms*, using generic clients (browsers) or virtual toolkits such as Java Swing or Tk. Those techniques do not provide any adaptation perceivable to the user – except some rendering differences between browsers and adaptation to the platform look-and-feel for some virtual toolkits [2] – and do not offer a satisfying solution when considering systems that will run on devices with very distinct input/output capabilities [17]. Xweb [8] produces UIs for several devices starting from a multi-modal description of the abstract UI. This system operates on specific XWEB servers and browsers tuned to the interactive capacities of particular platforms, which communicate thanks to an appropriate XTP protocol.

A third approach is the *development of one single description for the common part and additional descriptions for the platform specific sides of the UI*. This approach is an extension of the generic client approach described above. Examples are XML documents together with CSS or XSL style sheet, or one XML document with XSLT transformations to WML or XHTML [13,14]. UIs produced with this approach are consistent at the level of information if not at the level of their appearance. However, they still require to develop one style sheet or XSLT document per target platform. The last techniques belong to the model-based paradigm. Recent tools such as ARTStudio [16] and TERESA [11] have extended the scope of automatic interface design to *multiplatform generation*. Those tools make use of various descriptions, called models. Some models such as the task model and the domain model contain the interactive system specifications at a high abstraction level while other models such as the platform model and the interactor model act more as a knowledge base that will be reuse in different system design processes. Starting from these models, the tools are able to produce a set of platform specific UI, depending on the platforms it supports. This approach has the drawback to offer little control to the designer: when a system is completely automatic, the designer cannot choose how the tasks will be shared among presentation units, or which widgets will be used, or what layout will be given to the final interface. Some systems however allow user-defined parameters. On the other side, automatic design tools present the advantage of "specify once generate many", which means that they are able to generate several UI starting from one single specification. A slightly different approach is the *specification-based interface design with model-based tools*. Specification based MB-IDEs [9] provide powerful interface specification languages [13]. The modelling languages of these MB-IDEs allows to express models at different abstraction levels.

Covigo has developed a system for paginating content of Web page by pattern (e.g. every fifth <tr>) or by size (e.g. 1024 KB). Pagination is a technique to break a large body of content into multiple pages [7].

LiquidUI, an authoring tool, for the UIML language [12] is a good example of a specification based multiplatform MB-IDEs. An important drawback of this approach is that each UI has to be described with a platform-specific vocabulary, what is not very different from the development of a specific interface for each platform, except that the vocabularies are simple to learn (UIML claims to be usable by everyone and not only by computer scientists) and that some common elements can be factorized. A recent development around UIML is the Transformation-based Integrated Development Environment (TIDE) [1]. TIDE goes far beyond the specification approach. It uses four abstraction levels: a task model, a description of the UI using UIML [12] with a generic vocabulary that is common for a device family (e.g. desktop or WML), a UIML description with a platform specific vocabulary and the final UI. The tool supports the mappings between the abstraction levels (the task model is not included yet), letting the designer controlling them.

Another model-based transformational approach is the Scalable Web technique [17]. Scalable Web address the problem of device heterogeneity in Web development by allowing authors to build a device-independent presentation model at design time. This model is provided for the device with the largest screen size. The presentation model is then submitted to two kinds of adaptations: pagination of large presentations into smaller and simpler ones and control transformations. Layout transformations are also realized. This kind of transformational approach offers at the same time a guarantee of continuity between the system versions and an adaptation of the UI to the specific targets. However, the set of transformations rules provided seems very limited and no specific attention is attached to continuity issues.

3. THE GRACEFUL DEGRADATION APPROACH

Like [17], we argue that it must be possible to centre the design effort on one source interface (or "root interface"), designed for the less constraint platform and to apply a set of transformation rules to this source interface in order to produce specific interfaces targeted to more constraints platforms. The phrase *more constraints platforms* covers:

- Platforms whose screen has a lower resolution.
- Platforms whose screen has similar resolution, but where the objects included in the interface are to be larger or more distant (e.g. touch screen interfaces) or where a part of the display is used for other purposes (e.g. virtual keyboard).
- Platforms where less widgets are available, because they have reduced versions of the toolkit or simplified versions of the mark-up language for example.
- Platforms where some widgets are much less usable, for example because of the absence of a keyboard on some platforms.

As our transformation rules take as input an interface tailored for a large screen and produce smaller interfaces as output, we call the transformation process a *degradation*. As we want to produce highly usable interfaces adapted to the specific platforms while preserving the consistency between the versions, we qualify this degradation as a *graceful* one.

4. RULES FOR GRACEFUL DEGRADATION

Design by Graceful Degradation requires a set of transformation rules that will adapt the source interface to each target platform. Graceful Degradation rules (hereafter GD rules) have been identified by the observation of the user interfaces of a large number of applications running on several devices. Some applications were publicly available, other ones were developed in collaboration with our research center, such as an information system developed for emergency services in Belgian hospitals that runs on workstations, Pocket PCs and a wall display.

5. RULES TYPOLOGY

GD rules have been classified using the CAMELEON framework [3] abstraction levels. The CAMELEON framework is intended to support the development of context-sensitive user interfaces in a model-based approach. It describes models at four abstraction levels (fig.1) from the task specification until the running interface [3,15,16]:

- *Tasks and Concepts*: this level describes the interactive systems specifications in terms of the user tasks to be carried out and of the domain objects manipulated by these tasks.
- The *Abstract User Interface* (AUI) is an expression of the UI in terms of presentation units, independently of which interactors are available. A presentation unit is a presentation environment (e.g. a window or a panel) that support the execution of a set of logically connected tasks.
- The *Concrete User Interface* (CUI) is an expression of the UI in terms of abstracts interactors and their position. The concrete UI is still a mock-up working only in the development environment. It can be modified by the designer.
- The *Final User Interface* (FUI) is generated from a concrete UI expressed in source code of any programming language or mark-up language (e.g. Java or HTML). It can then be interpreted or compiled.

GD rules can be considered at the CUI level, at the AUI level and at the Tasks and Concepts level. The FUI does not concern the designer anymore.

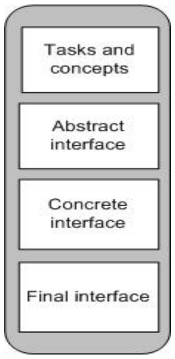

Figure 1 The four abstraction levels in the CAMELEON framework

6. GD RULES AT THE CONCRETE USER INTERFACE LEVEL

Two important kinds of GD rules can be applied at the Concrete User Interface level: rules that transform the layout relationships between the graphical objects and rules that modify the number and nature of the graphical objects.

6.1 Layout Relationships Transformations

There are three types of rules that can be applied to layout relationships: *resizing rules,* that modify the dimensions of a graphical object, *reorientation rules* that modify the orientation of an object without other change in size or position and *moving rules* that modify the localization of a graphical object, i.e. the position of the object in the containing window, either defined in the coordinates of this window, either in terms of constraints on the geometric relationships with the other objects (e.g. alignment, justification, ...). Theoretically, resizing rules could be applied to any UI component, but we have to take into account:

- The nature of the abstract interactor: some interactors have fixed dimensions in most of the toolkits where they have been implemented (e.g. a radio item) while others may generally be resized (e.g. a button).
- The constraints imposed by the toolkits: a lot of toolkits do not let the programmer give the widgets arbitrary dimensions: for example, widgets in languages like HTML or QTk give automatically the required size.
- The limits of human perception: for example, experimental usability results establish that an icon cannot be shrunk below the threshold of 8 x 7 pixels [6]. Beyond this, it becomes illegible or impossible to distinguish.

When a component can be resized, we have to know the minimum width and height it can be shrunk to (fig. 2). For some widget types, the minimum width / height of some interactors is influenced by factors that will only be determined at design time for a given application: e.g. the minimal width of a list box depends on the length of the larger proposed choice, the minimal width of a button depends on the length of its label, etc. When the aspect ratio should not necessarily be kept, the definition is enriched by the minimum height when minimum width is reached (fig. 3a) and the minimum width when the minimum height is reached (fig. 3b)

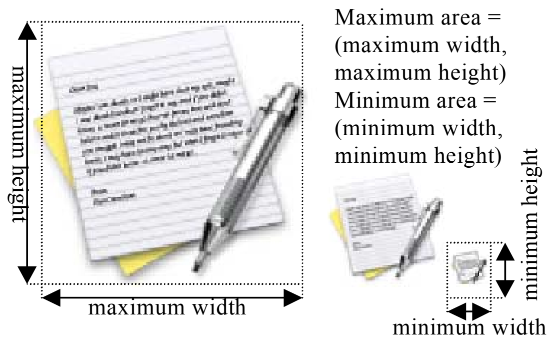
Maximum area = (maximum width, maximum height)
Minimum area = (minimum width, minimum height)
maximum height
maximum width
minimum height
minimum width

Figure 2 Minimum and maximum areas of component

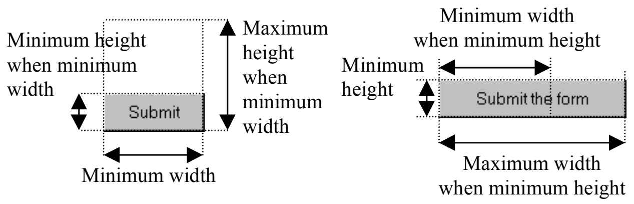
Minimum height when minimum width
Maximum height when minimum width
Submit
Minimum width
Minimum width when minimum height
Minimum height
Submit the form
Maximum width when minimum height

Figure 3 Minimum and maximum width and height

Reorientation rules are mainly useful when switching from landscape to portrait mode or conversely. They can only be applied to a small set of objects, such as table labels for example. Fig.4 shows an example of reorientation rule applied to an accumulator widget (i.e. an component transferring items from the left list of possible values to the right list of accumulate selected items).

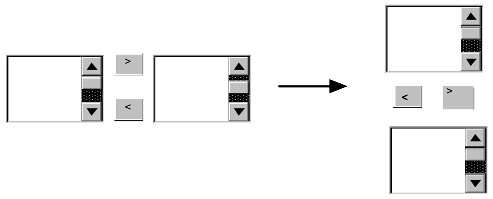

Figure 4 Reorientation rule

Moving rules are useful in several cases:

- The components do not fit in one dimension (horizontally or vertically) when there is blank space left in the other dimension.
- The components do not fit horizontally and we want to avoid horizontal scrolling.
- Some ergonomic rule or convention of the target platform has to be respected (e.g. menus on an IPaq should better be placed on the bottom of the screen).

6.2 Graphical Objects Transformations

Beside GD rules that transform the layout relationships between graphical objects, another type of transformation can be applied at the CUI level, namely modifications in the nature of the graphical objects (widgets, icons, windows,...). Object transformations can take three different forms: modification, substitution and removal.

Modification rules act upon the appearance of a graphical object. The physical rendering of a semantic feature can be modified (e.g. the notion of 'emergency' could be represented by the red colour on a workstation and by a flickering on a mobile phone), or the font of a text, or the colour of an object.

Substitution rules replace an interactor (i.e. an interactive graphical object, or widget in a GUI context) by an alternate interactor that enables the same type of functionalities. A substitution rule can be activated for two reasons:

- *Unavailability*: when an interactor is no longer available on the target platform, it has to be replaced by another one which is available on the target. For example, check boxes and radio buttons, non existing in WML language for mobile phones, are replaced by a list, as illustrated on fig.5.

-

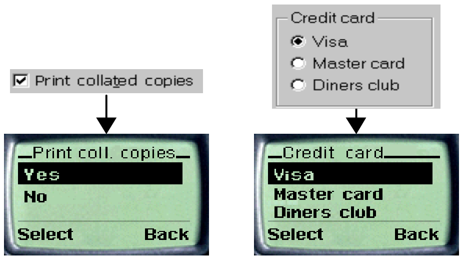

Figure 5 Component replacement due to unavailability

- *Screen size inadequacy*: if an interactor does not fit in the target platform because it takes too much screen size, it has to be replaced. For example, fig.6 shows possible substitutions for an accumulator, an interactor transferring items from the left list of possible values to the right list of accumulated selected items, thus allowing multiple selection among a closed list of items. The accumulator can be replaced with a smaller version of the same object (with the transfer buttons labels being reduced to the accumulator is no longer affordable, the use of other interactors supporting multiple selection tasks has to be considered: a group of check boxes, a list box containing check boxes, a simple list box or a list restricted to merely one item in the extreme case. Similarly, fig.7 shows a set of substitutions for a simple choice task.

Different types of substitution can be performed:

- *Simple substitution* (1→1): interactor x on the source platform is replaced by interactor y on the target platform.

- *Regrouping* (N→1): a set of interactors on the source platform is replaced by a single interactor on the target platform. E.g. a set of check buttons can be regrouped into an accumulator.

- *Splitting* (1→N): a single interactor on the source platform is replaced by a set of interactors on the target platform. E.g. a tabbed panel could be replaced by a set of hyperlinks.

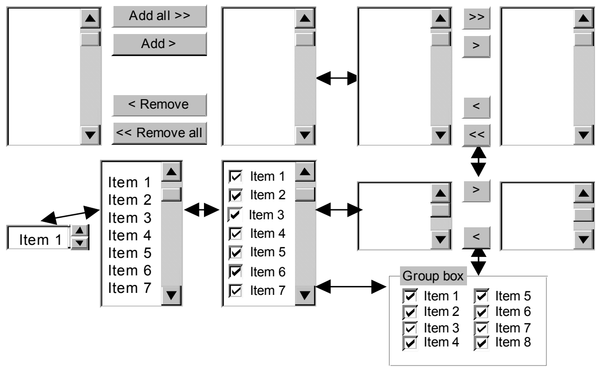

Figure 6 Candidate interactors for multiple choice

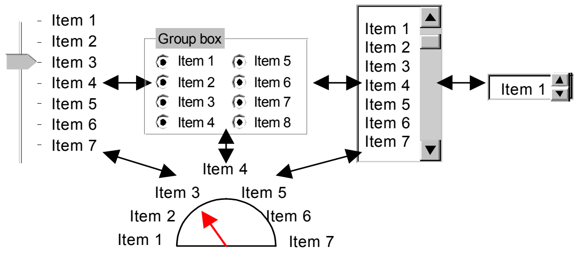

Figure 7 Candidate interactors for simple choice

Not all alternatives have the same ergonomic quality in a given context:

- Not all interactors are as easy to manipulate on a given platform: a check box is difficult to select on touch screen platforms because of the finger dimension.

- Some interactors offer a better visual guidance for a given type of task. For example, an accumulator clearly denotes a multiple selection task whereas a simple list box does not indicates whether multiple choice is allowed neither how to achieve this task. Only experienced users will know that they have to press a special key in order to select multiple items

- Depending on the number of available choices, some interactors seem to be more appropriate than others. For example, ergonomics rules generally state that a group of check boxes should be limited to 7 items in order to optimize the legibility, whereas an accumulator is perfectly suitable for higher cardinality.

Interactor substitution rules can also operate at a semantic level, taking into account domain characteristics.

Let us consider the choice of a month. This is a simple choice among a predefined set of twelve possible values ranging from January to December. The rules exhibited above apply: we could use one of the interactors shown in figure 7. However, since months have a special semantic meaning and are continuous over time (horizontality typically represents the time dimension), a dedicated series of substitutions may be used instead (fig. 8).

The last type of GD rule that can be applied to graphical objects are *removal rules*, that merely delete a graphical object, due to space constraints on the target platform (e.g. removal of pictures on a mobile phone).

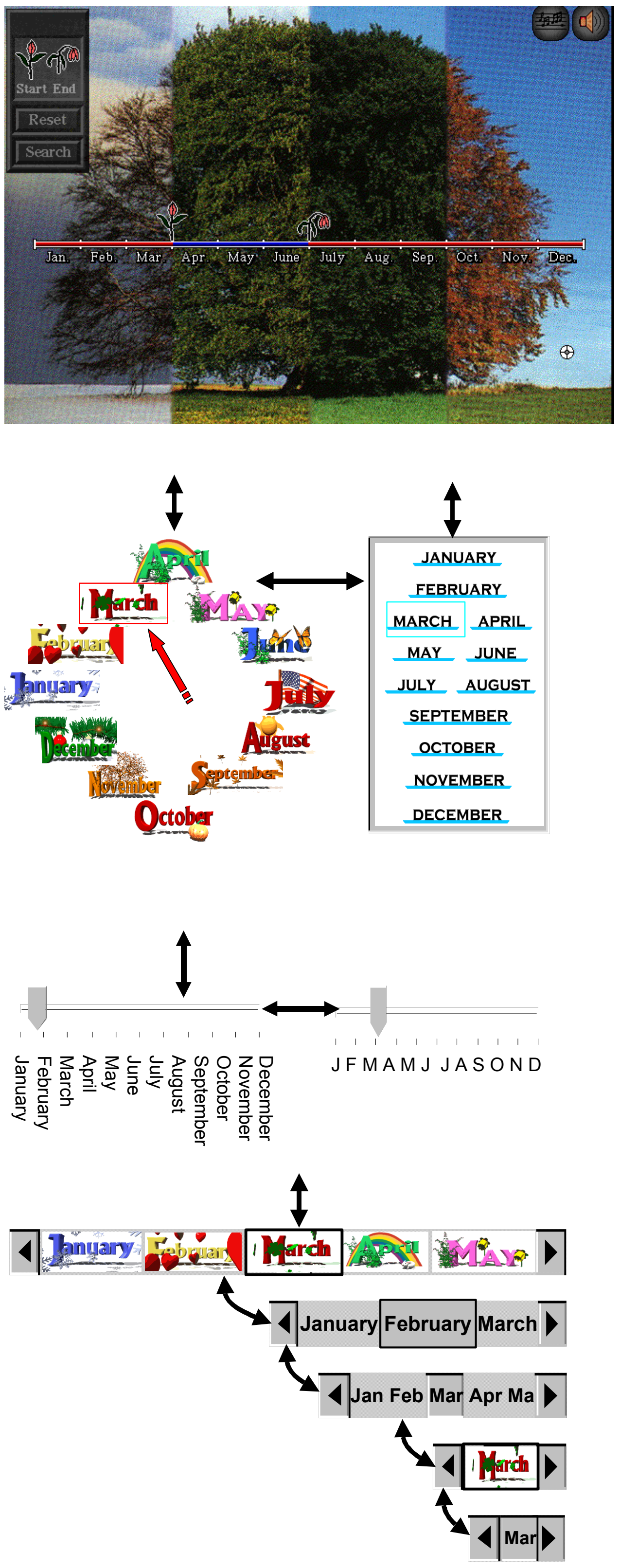

Figure 8 Semantic substitution rules for month selection

7. GD RULES AT THE AUI LEVEL

The AUI level defines the distribution of the interactive tasks among the presentation units. A *presentation unit* regroups low level tasks (e.g. selecting an item or entering text) that are logically linked together and that will be achieved within the same presentation (window, panel, dialog box, card, ...)

When there are big differences between the platforms constraints (e.g. big differences in screen size and resolution), it will not be possible to maintain a same distribution of tasks between the system versions. Fig.9 shows the distribution possibilities in platform specific versions of a system: *similar versions* are versions sharing the same task distribution among presentation units, when *distributed versions* allocate the same set of tasks differently from version to version.

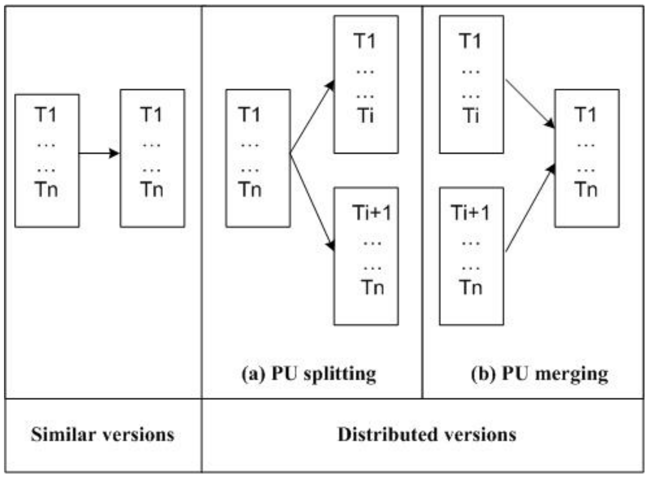

Figure 9 Distribution possibilities in platform specific versions of a system.

At the AUI level, the most useful GD rules split the source presentation unit into two or more presentation units on the target platform. We call these rules *splitting rules*. Conversely, merging of presentation units is also possible.

A possible side effect of the application of a splitting rule is the introduction of internal redundancy within distributed versions of a system: a task that appears once on the source platform could appear on two or more presentation units on the target platform. Fig.10 shows an example of internal redundancy caused by a platform change: the single "cancel" task on the source platform has to be duplicated on the target platform.

Beside splitting rules, another possible adaptation technique at the Abstract UI level is the *reorganization of tasks within the same presentation unit*. A reason for internal permutation between tasks can be that we want to present tasks in the order of frequency of each task and that we expect that a task frequency will change on the target platform (e.g. the consultation of an address book could be more frequent on a mobile phone than on a workstation).

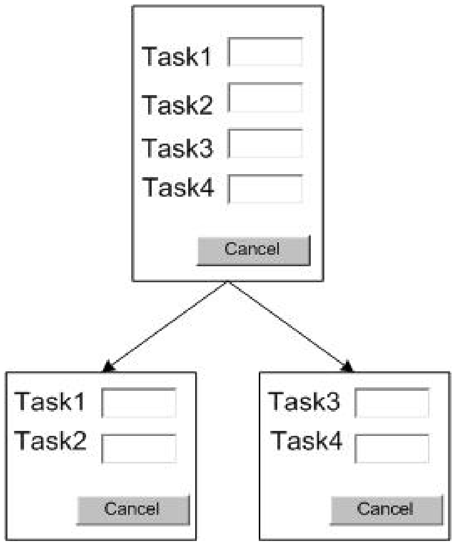

Figure 10 Internal redundancy due to splitting rule

8. GD RULES AT THE TASKS & CONCEPTS LEVEL

At the Tasks & Concepts level, GD rules can be applied to general functionalities (high level tasks, that correspond to the user's general goals), to the procedures that the user must follow in order to achieve his/her general goals (low level tasks), to the temporal ordering between tasks and to the concepts.

8.1 General Functionalities Transformations

There are two types of GD rules that modify the general functionalities of a system: high level task deletion and high level task insertion.

8.1.1 High Level Task Deletion

A high level task present on the source version may be removed from the target version, for different reasons:

- The task implies interaction capabilities that are unavailable or inappropriate on the target platform (e.g. tasks involving video streaming or manipulation of complex graphics are impossible to perform on a cellular phone, so are tasks of data storage when the quantity of data is huge).
- The task requires resources that are very scarce on the target platform so that the interaction could be interrupted due to lack of resources (e.g. a task manipulating an object requiring much RAM memory).
- The task requires such interaction capabilities that carrying out this task on the target platform could become very tedious (e.g. a task of word processing on a PDA, although partially possible, rapidly becomes impractical due to the limited entry capabilities like virtual keyboard or character-recognition).
- The typical context of use of the target version is inappropriate to the performance of that task (e.g. a task of graphical edition is inappropriate in a context where the user will be standing, when the target platform is an interactive kiosk for example)

8.1.2 High Level Task Insertion

A high level task not available on the source version may be added on the target version, for the same types of reasons (changes of interaction capabilities or changes of the typical context of use on the target platform).

8.2 Procedures Transformations

Another type of transformation at the Tasks & Concepts level affects the subtasks necessary to achieve the same general functionality. Two types of rules modify subtasks: subtasks deletion rules and subtasks insertion rules.

Subtasks Deletion

Subtasks can be deleted for different reasons:

- Some subtasks are unnecessary on the new platform (e.g. on a platform with a GPS system, it is not necessary anymore to specify the user's location).
- Some subtasks require too much resources with respect to the constraints of the target platform (e.g. the cellular phone version of an information system dedicated to theatre will still enable the general task of booking theatre tickets, but not anymore the subtask of viewing the free seats on a picture of the hall.

Subtasks Insertion

Causes for subtasks insertion involve:

- Insertion of a subtask because the target platform does not permit to execute several tasks at the same time (e.g. on a mobile phone, as it is impossible to edit several information items simultaneously, a selection task that would allow the user to choose which item he wants to modify should be added before any edition task mapped to more than one item)
- Insertion of a subtask because the display area on the target platform does not permit to execute the same set of tasks within one presentation only, so that the tasks have to be split among several presentations, what may imply the insertion of additional navigation tasks between the new presentation spaces.

Temporal Ordering Transformations

Examples of GD rules modifying the temporal ordering between tasks are:

- Sequentialization of tasks when the style of interaction changes (e.g. from a GUI to a speech-based conversational interface, or from a direct manipulation UI to a form-based UI)
- Conversely, some tasks that where sequential can become concurrent when the style of interaction changes

Concept Level Transformations

Graceful degradation rules can modify the view given on some concepts:

- information can be summarized or cut
- some attributes can be masked
- alternative shorter label or titles can be chosen
- etc.

GRACEFUL DEGRADATION RULES AND CONTINUITY

Not all GD rules that can be applied to a source interface have the same impact on the continuity within the multiplatform system. Intuitively, we think that rules applied at a lower level in our framework (e.g. resizing a graphical object) generate less discontinuity than rules applied at a higher level (e.g. high level task deletion).

Following this principle, we have established a priority ordering between graceful degradation rules. Rules with a high priority should be the first to be tried when adapting the source UI to a target platform. Rules with a lower priority should only be applied when higher priority rules have failed to transform the source UI into a UI that respects the target platform usability criteria. We propose the following list of graceful degradation rules, from the rules with the highest priority to the rules with the lowest priority:

- *Layout transformation* (modification of the layout relationships between graphical objects). The layout transformation rule that seems to introduce the less discontinuity is the resizing rule, then comes the reorientation rule, then the moving rule (fig.11)

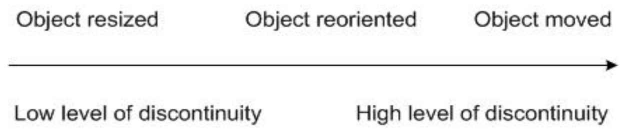

Figure 11 Level of discontinuity induced by layout transformations rules

- *Graphical object transformation*. The simple modification of the interactor's appearance do not cause a lot of discontinuity (e.g. color change, ...). The substitution of an interactor by another interactor supporting the same type of functionalities induces more discontinuity (e.g. substitution of an accumulator by a list box). More discontinuity is perceived if the substituted interactor has a different shape. Regrouping or splitting interactors creates still more discontinuity (e.g. substitution of a group of check boxes by a listbox or conversely). The highest level of discontinuity for graphical object transformation rules is achieved when deleting an interactor (see priority ordering on fig.12).

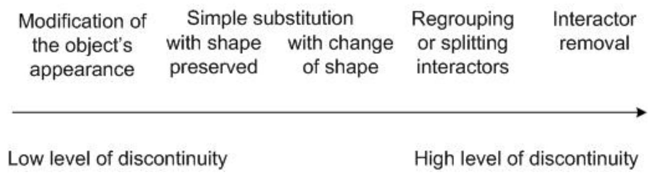
Modification of the object's appearance
Simple substitution
with shape preserved
with change of shape
Regrouping or splitting interactors
Interactor removal
Low level of discontinuity
High level of discontinuity

Figure 12 Level of discontinuity induced by graphical objects transformations rules

- *Tasks reorganization*. Two types of reorganization rules can be applied: reorganization within the same presentation unit or splitting rules that distribute the tasks belonging to a same presentation unit on the source UI between distinct presentation units on the target platform. Rules of the first type obviously generate less discontinuity than rules of the second type (fig.13)

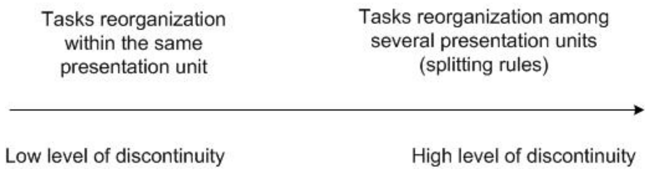
Tasks reorganization within the same presentation unit
Tasks reorganization among several presentation units (splitting rules)
Low level of discontinuity
High level of discontinuity

Figure 13 Level of discontinuity induced by task reorganization rules

- *Transformations at the Tasks and Concepts level*. These transformation rules generate important differences between the platform specific versions of the UI. We propose to give a higher priority to temporal ordering transformations rules, that preserve the displayed information and the available tasks. Concept level transformations and procedures transformations generate more discontinuity and should be given a lower priority. The lowest priority is given to general functionalities transformations rules, that significantly modify a system (fig.14).

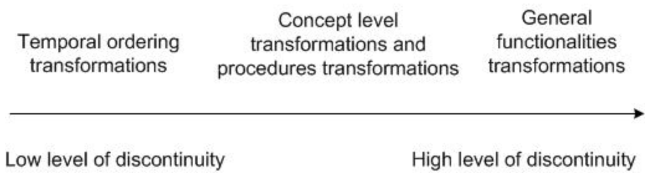
Temporal ordering transformations
Concept level transformations and procedures transformations
General functionalities transformations
Low level of discontinuity
High level of discontinuity

Figure 14 Level of discontinuity induced by transformation rules at the Tasks and Concepts level

The proposed priority ordering has still to be validated by usability studies conducted with end users. Both the performance and the preference of the users have to be recorded. The performance can be evaluated by the time required to perform a task on the source and target interface. The preference can be obtained by asking the users to classify several designs, where each design results from the application of a single different rule to the same source interface.

9. CONCLUSION AND FUTURE WORK

In this paper, we have introduced the notion of graceful degradation as method for designing multiplatform systems with a focus on continuity. The graceful degradation approach is based on an original set of rules. These rules are described and classified in a model-based framework. A priority ordering between rules is then proposed, that still has to be validated by empirical studies. Future work includes the formalization of some of the rules described above, with the aim of applying them automatically in two cases: systems able to adapt their user interface at run-time in response to changes in the screen resolution and a design environment that will provide designers with assistance in obtaining a graceful degradation of UIs.

10. ACKNOWLEDGMENTS

We gratefully acknowledge the support of the Salamandre Project, funded by the "Initiatives III" research program of the Ministry of Walloon Region, DGTRE, Belgium

(http://www.isys.ucl.ac.be/bchi/research/salamandre.htm).

11. REFERENCES

[1] Ali M.F., Pérez-Quiñones M.A. and Abrams M., Building Multi-Platform User Interfaces With UIML, in: A. Seffah & H. Javahery (eds.) *Multiple User Interfaces: Engineering and Application Framework.* John Wiley and Sons, 2003.

[2] Bickmore, T.W., Schilit, B.N. (1997), Digestor: Device-Independent Access to the World Wide Web, in *Proc. WWW'7 Conference*

[3] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J., *A Unifying Reference Framework for Multi-Target User Interfaces*, Interacting with Computers, V15N3, June 2003, 289-308.

[4] Denis C. and Karsenty L. Inter-usability of multi-device systems: A conceptual framework, in: A. Seffah & H. Javahery (eds.) *Multiple User Interfaces: Engineering and Application Framework*, John Wiley & Sons, 2003.

[5] Elting, Ch., Zwickel, J., Malaka, R., Device-Dependent Modality Selection for User Interfaces – An Empirical Study, in *Proceedings of 6th Int. Conf. on Intelligent User Interfaces IUI'2002* (January 13-16, 2002, San Francisco), ACM Press, New York, 2002

[6] Kamba, T., Elson, S.A., Harpold, T., Stamper, T., Sukaviriya, P.N., Using Small Screen Space More Efficiently, *Proceedings of ACM Conf. on Human Aspects in Computing Systems CHI'96* (Vancouver, 13-18 April 1996), ACM Press, New York, 1996, pp. 383-390.

[7] Mandyam, S., Vedati, K., Kuo, C., Wang, W., User Interface Adaptations: Indispensible for Single Authoring, in *Proceedings of W3C Workshop on Device Independent Authoring Techniques* (St. Leon-Rot, 15-26 September 2002)

[8] Olsen, D.R., Jefferies, S., Nielsen, T., Moyes,P., Fredrickson, P., Cross Modal Interaction using XWEB, in *Proceedings of the 13th annual ACM symposium on User interface software and technology UIST 2000* (San Diego, United States), ACM Press, New York, pp. 191-200.

[9] Paternò, F., *Model-Based Design and Evaluation of Interactive Applications*, Springer-Verlag, Berlin, 2000.

[10] Paternò F. and Santoro C., One Model, Many Interfaces, in *Proceedings of CADUI 2002, the 4th International Conference on Computer-Aided Design of User Interfaces* (Valenciennes, France, May 2002), 143-154.

[11] Paternò, F., Mori, G., Santoro, C., Tool Support for Designing Nomadic Applications, in *Proceedings of 7th Int. Conf. on Intelligent User Interfaces IUI'03* (January 12-15, 2003, Miami), ACM Press, New York, 2003

[12] Phanouriou C., UIML : *A Device-Independent User Interface Markup Language*. Ph. D. Thesis, Virginia University, 2000.

[13] Puerta, A., Eisenstein, J. (1999), Towards a General Computational Framework for Model-Based Interface Development Systems Model-Based Interfaces, *Proceedings of 3rd Int. ACM Conf. on Intelligent User Interfaces IUI'99* (Redondo Beach, 5-8 January 1999), ACM Press, New York, pp. 171-178, accessible at http://www.arpuerta. com/pubs/iui99.htm

[14] Puerta, A., Eisenstein, J. (2003), Developing a Multiple User Interface Representation Framework for Industry, in: A. Seffah & H. Javahery (eds.) *Multiple User Interfaces: Engineering and Application Framework.* Wiley and Sons, 2003.

[15] Thevenin, D., Coutaz, J. (1999), Plasticity of User Interfaces: Framework and Research Agenda, *Proceedings of 7th IFIP Int. Conf. on Human-Computer Interation INTERACT'99* (Edinburgh, 30 August-3 September 1999), IOS Press, Amsterdam, pp.110-117, accessible at http://research.nii.ac.jp/~thevenin/papiers/Interact9 9/Plasticity.Interact99-WWW.pdf

[16] Thevenin D., *Adaptation in Human Computer Interaction: the case of Plasticity*. Ph. D. Thesis, Joseph Fourier University, Grenoble, 2001.

[17] Wong C., Chu H.H. and Katagiri M. A Single-Authoring Technique for Building Device-Independent Presentations, in *Proceedings of W3C Workshop on Device Independent Authoring Techniques* (St. Leon-Rot, 15-26 September 2002), accessible at http://www.w3.org/2002/07/DIAT/posn/docomo.pdf