

## **An Analysis of Graceful Degradation as a Design Method for Multi-Platform User Interfaces**

### ***Introduction***

People increasingly use a variety of devices – PCs, PDAs, mobile phones, etc. - to access the same applications. The widespread use of a variety of computing devices has posed some new challenges to design of user interfaces. Instead of designing applications solely for the standard PC, many designers now need to work across a variety of platforms that vary greatly in capabilities such as screen resolution, mobility, input and output methods, and so forth.

Some current methods for developing UIs for multiple platforms include:

- **Designing a specific interface for each platform.** This has the advantage of giving the designer control over the different versions. However, designing several interfaces is time-consuming. Coordination of designs across different platforms can be difficult and error-prone, especially if different designers are assigned to work on the different versions. (Lin and Landay) Furthermore, this technique does not guarantee a continuity of user experience across platforms. (Florins and Vanderdonct, 2004)
- **Designing one generic interface, and use a special-purpose program to generate the other versions.** This approach can save time and effort. However, designers have little control over the final products. The special-purpose programs rarely work as desired, and they lead to interfaces that are awkward to use. (Lin and Landay)

Several researchers have proposed model-based approaches for developing multi-platform applications. In model-based approaches, the user interface is specified as a number of tasks and relations, instead of as a set of abstract or concrete interactors. In one such approach (Mori et al, 2003), designers develop a single model of high-level tasks, and then map the tasks for each specific platform. From the task model, user interfaces

are generated for each targeted platform.

In contrast, Florins and Vanderdonck have proposed that a set of graceful degradation rules can be used as a method for generating consistent user interfaces across multiple platforms (Florins and Vanderdonck, 2004). Graceful degradation (GD) is a principle that allows a system to continue to operate in the event of a failure or fault in one or more of its components. A user interface designed with GD as a principle allows all targeted platforms to display the interface, with a degraded display or reduced functionality on some platforms. This paper analyzes the proposed graceful degradation method for two specific cases:

- Generating user interfaces for mobile devices, using the device capabilities described in WURFL (an XML document listing all device capabilities) as a basis for degradation.
- Generating user interfaces for a multi-platform application where the tasks users want to accomplish vary according to the capabilities of the device and the context of its use.

Florins and Vanderdonck acknowledge that the use of the GD rules for transforming interfaces has not been validated by usability studies. This paper also does not examine the usability of the resulting generated interfaces, but instead focuses on the feasibility and appropriateness of using the GD method in the cases described above.

### ***Graceful Degradation as a Method for Multi-Platform UI Design***

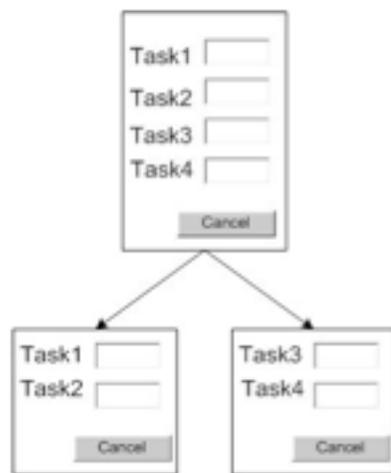
As stated above, graceful degradation is a principle that allows a system to continue to operate in the event of a failure or fault in one or more of its components. When a system is designed to gracefully degrade, whenever it encounters a fault, it continues to operate at a reduced level, instead of failing completely. GD is an important principle in web design. A site designed with this principle takes advantage of the latest features in newer browsers, while still allowing older browsers to access its content, though with reduced functionality or less desirable presentation qualities.

Florins and Vanderdonck propose that GD can also be used as a method for designing how applications can work across different devices (Florins and

Vanderdonckt). Since each device has different constraints, the authors propose that an application can be designed to target the device with the fewest constraints (i.e. the PC), with planned degradations for more constrained devices (i.e. PDA and mobile phone). With this approach, consistency of the UI across the different devices is a guiding principle, as “users expect to be able to reuse their knowledge of a given version of the system when using the same service on another platform.”

The authors propose a set of transformation rules governing GD across platforms, while maintaining a maximum level of consistency. GD rules are classified according to the CAMELEON abstraction levels, a framework for modeling interfaces, intended to support the development of user interfaces across multiple platforms. (Calvary, et al.) Different GD rules can be applied at each level.

- **Tasks and Concepts (TC).** This level describes high-level tasks that correspond to user goals. Rules that delete, modify or re-order tasks can be applied here.
- **Abstract User Interface (AUI).** At this level, tasks are grouped into logical presentation units. When platforms differ greatly in screen resolution, it may not be possible to maintain the same presentation units across all platforms. Rules that split or reorganize presentation units can be applied.



*Figure 1 - Splitting of presentation unit* (Florins, et al, p. 145)

- **Concrete User Interface (CUI).** Describes the UI in terms of interactor types and layout. At this level, rules that transform layout, or that modify the nature or number of graphical objects can be applied. For example, a component can

be resized to minimum dimensions, or reoriented. Interactor types can also be substituted if certain types are not available on more constrained platforms.

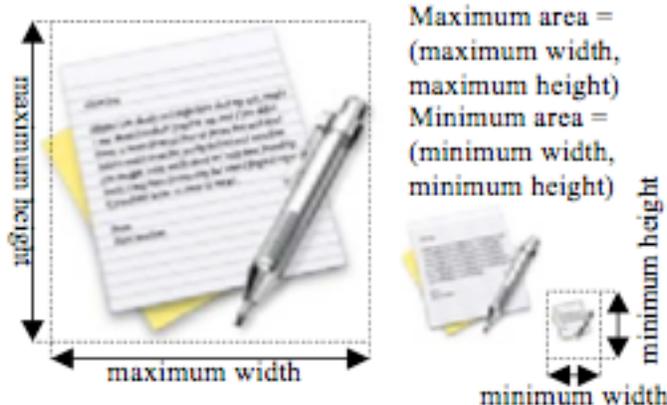


Figure 2 - Resizing components (Florins, et al, p. 142)

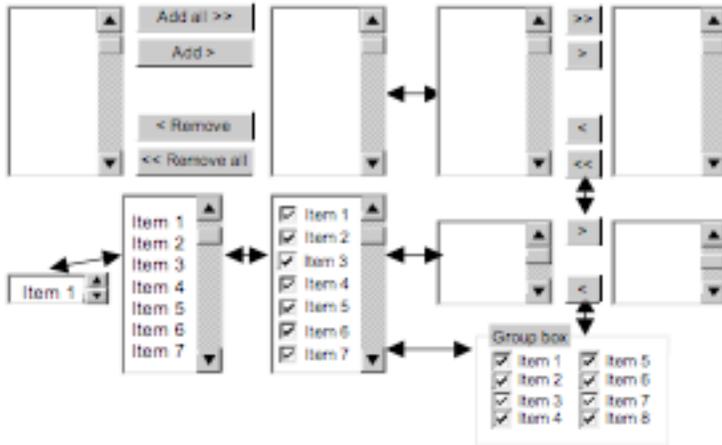


Figure 3 – Different interactors for multiple choice (Florins, et al, p. 143)

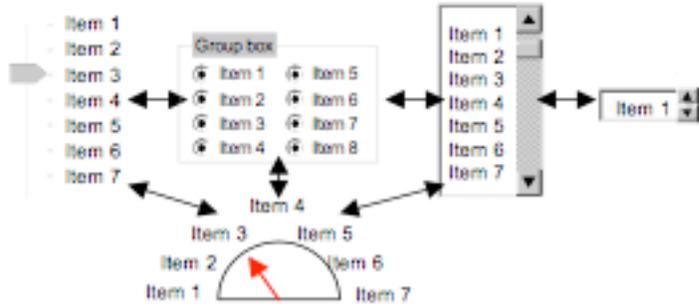
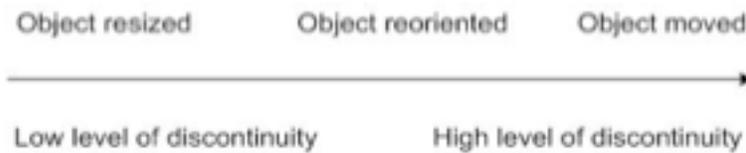


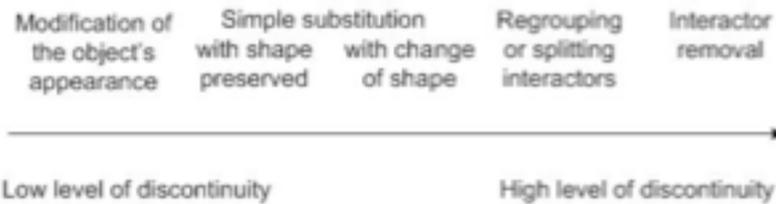
Figure 4 – Different interactors for single choice (Florins, et al, p. 143)

- **Final User Interface (FUI).** The FUI is generated from the Concrete UI, and can be compiled as UIs for different platforms. No GD rules can be applied here.

Rules are prioritized to maximize continuity across platforms, so lower-level GD rules are applied before higher-level rules. For example, resizing a graphical object is less disruptive than removing a task, so the re-sizing rule would be applied before resorting to the task deletion rule. CUI rules are applied before AUI rules, which are in turn applied before TC rules.



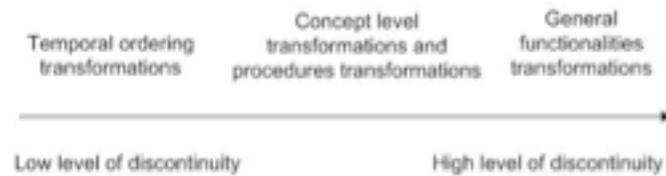
*Figure 5 – Level of discontinuity introduced by layout transformation rules at the CUI level (Florins, et al, p. 146)*



*Figure 6 – Level of discontinuity introduced by graphical object transformation rules at the CUI level (Florins, et al, p. 146)*



*Figure 7 – Level of discontinuity introduced by task reorganization rules at the AUI level (Florins, et al, p. 146)*



*Figure 8 – Level of discontinuity introduced by transformation rules at the TC level*  
 (Florins, et al, p. 146)

### ***Comparing Design-time vs Run-time Graceful Degradation***

Although Florins and Vanderdonckt proposed the GD rules for use in single-authoring user interfaces for multiple devices, the GD method could also be used to support consistency when manually designing separate interfaces for multiple devices. As mentioned above, different designers are often assigned to work on different versions of the interface for different devices, so a governing set of rules would be useful in promoting consistency across all versions.

When using the GD method during both design-time and run-time, designers first need to model the application according to the CAMELEON abstraction levels: Task and Concepts, Abstract User Interface, and Concrete User Interface. Designers then determine the least constrained platform, and design a source interface.

Using the GD rules as a set of design rules, designers would choose a subset of more constrained devices that represents the total set of target devices. For this subset, designers would apply the GD rules, according to the priority ordering, to the source interface in order to create different versions of the interface. For the new versions of the interface, designers will likely make adjustments in order to improve continuity and usability, likely overriding the GD rules in some cases.

When using the GD method for run-time degradations, programmers would need to write transformations to degrade source interface according to the GD rules, and then write scripts to run the transformations according to the GD priority rules. At run time, the application would detect the device and run the appropriate scripts, which would then transcode the source interface into an interface that the device can process.

When an application runs on many platforms, using GD as a method for single-authoring is more efficient than using it during design-time. Manually designing user interfaces using GD as a set of guiding rules is less efficient, but it may allow designers more flexibility to make changes to the interface for better usability. However, in cases where there are too many devices to realistically design an interface for each one, the GD method may provide a good solution for generating usable interfaces. The remainder of this paper assumes a single-authoring process, using GD during run-time.

## ***WURFL-based Graceful Degradation for Mobile User Interfaces***

One of the main challenges in developing applications for the mobile phone is the large number of phones and the varying capabilities of each phone. Ideally, a mobile application should work on all mobile phones. The WAP (wireless application protocol) standard was created by the major players in the telephony industry to address this issue. While all device manufacturers and network operators supposedly support the WAP standard, in practice phone and browser manufacturers each interpret and apply the WAP standards differently. The wide variety of device capabilities makes it hard for developers to build WAP and Wireless services which work acceptably well for all devices. Essentially, each device can be considered a different platform.

### **About WURFL**

To address these differences, the mobile developer community has created WURFL, an open-source, community-maintained XML file that describes the capabilities of all devices. WURFL is based on the concept that all devices are descendants of a generic device, but they may also descend from more specialized families. For example, all phones in the Nokia 7110 series descend from a generic Nokia phone, but the sub-models vary in their support of WML tables. WURFL has a mechanism called *fall\_back*, which is similar to the concept of inheritance in object-oriented programming. *fall\_back* lets programmers derive the capabilities of a phone from its family, unless a feature is different for that specific phone. Because of the use of inheritance and *fall\_back*, WURFL does not need to maintain a large matrix of all devices and all capabilities.

## Graceful Degradation with WURFL

In analyzing how we can use the GD method described above for designing interfaces across all mobile phones, we first look at the capabilities described by WURFL. WURFL lists capabilities found on devices in general, as well as capabilities associated with specific browser types: WML, CHTML, and XHTML. WML (Wireless Markup Language) is an XML-based format for specifying content on mobile devices that use WAP. WML precedes CHTML and XHTML. CHTML (Compact HTML) is a subset of HTML, designed for use with small information devices. Finally, XHTML (Extensible HTML) is a reformulation of HTML as an application of XML. Mobile device browsers commonly use one of these three mark-up languages.

The table below lists the display-related capabilities listed in WURFL. Other than browser-related, Java and WAP capabilities, these are the capabilities most relevant for user interfaces.

### *WURFL display-related capabilities*

<b>resolution_width</b>	Display's usable width expressed in pixels
<b>resolution_height</b>	Display's usable height width expressed in pixels
<b>columns</b>	Number of columns presented
<b>rows</b>	Number of lines presented
<b>max_image_width</b>	Width of the images viewable width expressed in pixels
<b>max_image_height</b>	Height of the images viewable width expressed in pixels
<b>Colors</b>	The number of colors used by the phone

The display-related capabilities can be used to apply some of the GD transformation rules described above. An application can be designed to target the device with the largest screen resolution and largest range of color display, and then degraded for more constrained devices. *resolution\_width* can be used to determine if a graphical object should be resized. *columns* and *rows* can be used to determine if a presentation unit should be split. For GD rules around layout transformation or substitution of interactor types, we next look at browser-related capabilities.

In comparing capabilities across different browsers, we find there is very little overlap. (See *Appendix A* for a comparison of WURFL capabilities across browser types) Some examples include:

- CHTML and WML browsers have a single capability about whether or not tables are supported. XHTML browsers have capabilities for table content layout, coloring table cells, and including form elements in tables.
- CHTML browsers can support "emoji" - special characters which appear in i-Mode (a wireless Internet service popular in Japan) as small icons - but WML and XHTML browsers don't know anything about emoji.
- WML browsers have several capabilities dealing with menu items, such as listing numbers with menu items and supporting icons as links. CHTML and XHTML browsers don't have any menu or list capabilities.
- XHTML browsers have several capabilities dealing with selection, such as whether or not radio buttons and dropdown lists are supported. CHTML and WML browsers don't have any capabilities around selection.

The GD method starts with a design for the least constrained device. With so little overlap in browser capabilities, it is difficult to determine which device is the least constrained, or which browser the most capable. Each browser has some capabilities the others do not.

One possible approach would be to combine all WML, CHTML and XHTML browser capabilities, design an interface for a hypothetical abstract browser, and begin degradation from there. However, some of the capabilities may conflict or be incompatible. For example, the XHTML capabilities around selection (*xhtml\_select\_as\_dropdown*, *xhtml\_select\_as\_popup*, etc.) may conflict with the WML capability *elective\_forms\_recommended* (input and select elements can/should be placed in a single card rather than on discrete cards). In the case of a conflict, it would be difficult to determine which capability to include in a design for the most capable browser.

The concept of the abstract browser is analogous to the abstract class in object-oriented programming. Like the abstract browser, an abstract class cannot be instantiated. Instead, sub-classes are derived from the abstract class, using some methods as defined by the abstract class, and overwriting other methods. Similarly, we could design a user interface for the abstract browser that includes all capabilities, even those that conflict, knowing that our design targets no realistic device. All devices would display a sub-

version of the abstract user interface, and WURFL capabilities could be used to generate the sub-versions. For example, devices with WML capabilities would ignore interactors relevant to CHTML and XHTML browsers, and display only portions of the interface appropriate for WML devices.

The proposed GD rules allow for deletion of tasks at the Tasks and Concepts level, but does not allow for removal of interactors at the Concrete User Interface level. Since the abstract design includes capabilities for all browsers, some interactors will necessarily be irrelevant. Removing or ignoring these interactors would fall outside the proposed GD rules.

Another approach would be to do three designs; one each for the least constrained WML, CHTML and XHTML browser. In addition to targeting the device with the least constrained display capabilities, each design would include all the WML (or CHTML or XHTML) capabilities described in WURFL. We would then use GD to generate designs for devices with fewer browser capabilities. For example, a design for the least constrained XHTML device could use a dropdown as a selection interactor. For XHTML devices without the *xhtml\_select\_as\_dropdown* capability, we may use radio buttons as a replacement selection interactor.

This second approach is a combination of using the GD method along with the existing method of designing a specific interface for each platform. We would still need to design three separate versions for WML, CHTML and XHTML devices, and we would still need to coordinate design changes across these separate versions. However, we would be able use the GD method to handle the many differences in capabilities within the same browser types.

## ***Graceful Degradation for Highly Adaptive Multi-platform Applications***

In the case described above, using WURFL capabilities as a basis for GD across all mobile devices, we assumed that an application would ideally have the same functionality across all devices. (At the Task and Concepts level, tasks may be deleted; but this is a last resort GD rule, as removing a task disrupts the consistency of the user experience across devices.) In our next case, we consider GD as a method for generating

user interfaces for a multi-platform application where the tasks users want to accomplish vary according to the capabilities of the device and the context of its use.

When considering applications that work on all devices - not just mobile phones, but PCs, tablets, PDAs, etc. - capabilities vary so greatly in terms of screen resolution, mobility, input/output devices, etc. that it makes sense for users to expect varying functionality on the different devices. A device's restrictions and capabilities determine how users want to use that device.

Varying functionality across different devices may not be adequately handled by rule-based graceful degradation. Graceful degradation necessarily implies a reduced level of functionality for less capable devices. The mobile phone, though having a smaller screen resolution and than a PC, has many capabilities a PC does not - mobility, GPS functionality, SMS capabilities, etc. On an application that takes full advantage of a device's capabilities, the mobile user interface is not a degraded version of the PC user interface. Instead, it takes full advantage of the mobile device's own capabilities.

### **MANNA (Map Annotation Assistant)**

MANNA (Map Annotation Assistant) is a hypothetical application used by geologists, engineers and military personal to create annotated maps. (Eisenstein, Vanderdonckt, Puerta). It provides online collaboration and runs on several platforms. In one scenario, a geologist is going to a remote location in Northern California to examine the effects from an earthquake. From a desktop workstation, the geologist can use MANNA to download maps and reports. Once she's on the road, she uses a mobile phone, and MANNA has different functionality. Because of the limited screen resolution of the mobile phone, MANNA does not display regional maps. Instead, it displays driving directions to the earthquake zone from the geologist's current GPS position. Upon arriving at the site, the geologist uses a PDA to enter notes. Since the PDA requires a stylus for input, MANNA's UI on the palmtop does not require double-clicks or right-clicks.

To design such a highly adaptive, multi-platform user interface, the authors propose using MIMIC, a UI modeling language (Puerta, 1996). Three model components are relevant here:

- **Platform model.** Describes various computer systems that will run the UI. The model contains a separate element for each platform, with attributes describing the platforms features and constraints.
- **Presentation model.** Describes visual appearance of the UI, including hierarchy of windows and widgets. Each widget is modeled as an abstract interaction object, which is platform neutral. Each bstract interaction object is then associated with one or more concrete interaction objects, which are specific to a particular platform.
- **Task model.** A structured representation of the tasks users may want to perform. Tasks are divided into sub-tasks.

To handle varying functionality between different platforms, designers create mappings between the task and the platform elements. In turn, the task elements are mapped to presentation models.

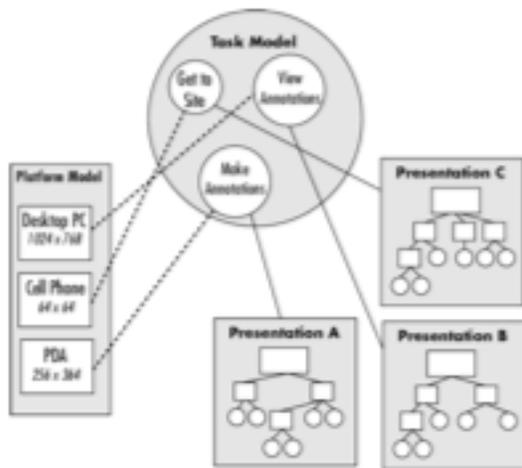


Figure 9 – Tasks elements mapped to platform elements and presentation models (Eisenstein, et al, p. 74)

In attempting to use the GD method here for generating interfaces across multiple platforms, we again face the issue of deciding which is the least constrained platform. As mentioned above, each platform has different capabilities, which makes it appropriate for different sets of tasks. Furthermore, the priority of tasks will vary across devices, depending on the most likely context of its use. For example, when using MANNA on a mobile phone, clicking on a spot on a map displays driving directions to that particular spot. When using MANNA on a PDA, clicking on a spot on a map allows users to enter

notes about that spot. On a PDA, driving directions are secondary, and are displayed only after a subsequent click. On a PC, clicking on a spot on a map displays detailed information about the selected area, as well as any previously entered notes. Similar to a PDA, driving directions are secondary, and are displayed only after a subsequent click. In this case, there is no least constrained overall platform. In addition, it would not be possible to design a single user interface for use in graceful degradation, which would reflect the differences in task priorities across various devices.

However, it may be possible to use the GD method in a more limited fashion, in combination with the MIMIC modeling components. As mentioned above, the varying functionality between different platforms are represented by mappings between the task and the platform elements, and mappings between the task elements and the presentation models. Instead of using GD to generate the entire interface, the GD method could be used to generate portions of the interface for tasks that are mapped to multiple platforms.

In the example discussed above, the mobile phone, PDA and PC can all display directions to a particular spot. For this specific task, the PC is the least constrained device. We can therefore design a direction display layout with interactors for the PC, and use our GD rules (substituting interactor types, splitting presentation units) to generate direction display layouts for the PDA and the mobile phone.

## ***Conclusion***

Florins and Vanderdonckt's proposed set of graceful degradation rules for generating user interfaces across multiple platforms work best when the following two conditions are true:

- **There is one platform that is the least constrained platform.** Once we have determined the least constrained platform, we can design an interface for this platform, and use GD rules to generate user interfaces for more constrained platforms.
- **By design, tasks are identical across various platforms.** GD rules are designed to preserve continuity across all platforms, so that users can reuse their knowledge of a given version of the system when using the same service on another platform.

When applications are designed so that different platforms support different sets of tasks, the GD goal of continuity is no longer relevant in the same way.

As more applications are designed to work on multiple platforms, it is unlikely that there will be one platform that is the least constrained, or that tasks will be identical across various platforms. Braun et al (Braun, Hartl, Kangasharji, Muhlhauser) discuss the “third age” of computing, where ubiquitous computing is prevalent, and users routinely use multiple devices simultaneously. The authors posit that most user interfaces will someday “span multiple devices, which work together to render the UI through several channels and modalities concurrently.” Instead of cramming a large amount of UI onto a low-resolution device, or reducing the amount of UI to conform to the device’s limitations - which is basically what graceful degradation seeks to do - Braun et al discuss applications whose UIs “overflow” onto the devices around them; they dynamically distribute portions of their interfaces to whichever devices are currently available to the user. For example, when a large screen and PDA is available, the UI for an audio player, “which consists of control buttons (play, pause, etc.), a play list, and a long text about the artist,” could be distributed so that the control buttons are on the PDA, while the play list and artist information are displayed on the large screen.

For such highly adaptive applications, the GD rules are not sufficient for generating entire user interfaces across multiple platforms. However, the GD rules may still be useful for generating portions of the interface for tasks that span different devices with varying capabilities.

## Sources

Ali, Mir Farooq, Perez-Quinones, Manuel A. *Using Task Models to Generate Multi-Platform User Interfaces While Ensuring Usability*, CHI 2002.

Braun, Elmar, Hartl, Andreas, Kangasharju, Jussi, Mulhauser, Max. *Single Authoring for Multi-Device Interfaces*, Telecooperation Group, Department of Computer Science, Darmstadt University of Technology.

Calvary, Gaele, Coutaz, Joelle, Thevenin, David, Limbourg, Quentin, Bouillon, Laurent, Vanderdonct, Jean. *A Unifying Reference Framework for multi-target user interfaces*.

Eisenstein, Vanderdonct, Jean, Puerta, Angel R. *Applying Model-Based Techniques to the Development of Uis for Mobile Computers*. (2001). ACM 1-58113-325-1/01/0001

Florins, Murielle, Vanderdonct, Jean. *Graceful Degradation of User Interfaces as a Design Method for Multiplatform Systems*. (2004) ACM 1-58113-815-6/04/0001

Florins, Murielle, Simarro, Francisco Montero, Vanderdonct, Jean, Michotte, Benjamin. *Splitting Rules for Graceful Degradation of User Interfaces*. (2006) ACM 1-59593-287-9/06/0001.

Gertini, Enrico, Santucci Giuseppe. *Modelling internet based applications for designing multi-device adaptive interfaces*. (2004) ACM 1-58113-867-9/04/0500.

Mori, Giulio, Fabio, Paterno, Santoro, Carmen. *Tool Support for Designing Nomadic Applications*. (2003) ACM 1-58113-586-6/03/0001.

Puerta, Angel R., Cheng, Eric, Tunhow, Ou, Min, Justin. *MOBILE: User-Centered Interface Building*, CHI 1999.

WURFL: <http://wurfl.sourceforge.net/index.php>

WALL: <http://wurfl.sourceforge.net/java/tutorial.php>