INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Development Frameworks for Mobile/Wireless User Interfaces: A Comparative Study

Simona Pestina

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

March 2002

ABSTRACT

Development Frameworks for Mobile/Wireless User Interfaces:
A Comparative Study

Simona Pestina

In this research, we investigated the development frameworks for mobile/wireless Web applications with a special consideration to the user interface (UI) component. The Web applications running on small mobile devices cannot support the full range of the traditional desktop Web functionality. This is due to the devices' physical and computational constraints. We described how the mobile devices' constraints and their increased diversity and connectivity are affecting the traditional UI development tools, methods and concepts. We also analyzed the development impacts that are emerging from the necessity of building consistent user interfaces for different mobile devices and platforms. In particular, we discussed how the current development frameworks could satisfy the constrained mobile environment while preserving the cross-platform usability. We conducted a comparative survey of these frameworks using different criteria such as data processing, data interactivity and presentation techniques, data transfer, security support, cross-platform or device specific availability, scalability, UI modeling techniques, UI delivery mechanism. We exposed the strengths and weaknesses of each framework while highlighting the areas for future improvements. Our survey highlighted the need for a "universal" high-level framework for mobile UIs development.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. CHAPTER I – Introduction to Mobile Computing Development

Both the wireless data market and the Internet are growing very quickly and are continuously reaching new customers. More and more people are accessing the Internet and the remote applications, through cellular phones and handheld devices. The Web-enabled applications contribute significantly to the success of PDA (Personal Digital Analyzer) and mobile telephone. Such applications allow a user to access to Web services and information. In the future more and more Internet access will be done through various mobile and wireless devices. Devices could be mobile because they are carried around by users (a PDA or a wearable computer), because they move themselves (robots!) or because they are embedded within some other moving object (a car computer). All these devices have very different capabilities and features (e.g. display size and resolution, bandwidth, processing power, input/output).

From another perspective, the Internet and e-business development is confronted with an increase growth in user variety, with a rapidly evolving hardware and software technologies increasing the variety of client devices and network channels used to access the server-side applications. The wireless applications are tailored to cell phones with mini-browsers, and PDA devices that can connect wirelessly to the Internet. In the same time the modem-equipped PDAs and other mobile computing devices could use a high-speed wired connection to download content from the Internet.

Wireless connectivity [8] enables business to operate faster, better, more cost-effectively and more profitably through the use of always on, always connected and always available content and applications. The mobile computing devices' functionality is mainly in the domains of m-commerce, location-based services, group communication, voice-and text based communication, storage and retrieval of personal data. This functionality have gone beyond the one-way delivery of weather updates, news headlines, and stock quotes to offer more interactivity, such as access to bank accounts, travel reservations, and email.

1.1. Mobile Computing

Mobile computing has emerged from convergence of two rapidly evolving network technologies, wireless data and the Internet. Mobile wireless computing offers many benefits. Corporations can provide workers with nonstop access to e-mail and enterprise applications, allowing them to work productively no matter where they are. Field data can be captured in real time and made available without delay to the rest of the organization, improving coordination and responsiveness. However, mobile computing is treated as a separate form of computing, posing *significant challenges*. It's a new type of computing, with *unique user behaviors* and *expectations*. The design of the user interface is governed by a separate set of design and usability principles including the need to make the user interface more context-oriented.

Mobile computing and the development in wireless communication have given users the expectation that they can access information and services whenever and wherever. In order to support this, the user interface needs to be adapted to a wide range of possible user situations. The complete environment in which the user is carrying out an interactive task determines the context of use. Two types of characteristics simultaneously determine the context of use:

- *Internal changes* that are related to the application and its UI (e.g., the computing platform, the software/hardware parameters, the interaction devices, the network bandwidth, the latency, connectivity, the screen resolution and other display capabilities).
- *External changes* that are independent from the system (e.g., the type of user, his role, skills, knowledge, preferences, his location, the stress level, the organization structure, the information channels, the sound and light conditions).

Any change of at least one of the above characteristics may generate a possible change of the context of use, endangering the predicted usability of a predefined UI. The internal changes may occur also within the same computing platform. According to Dey and Abowd in [11] the context represents the:

- *Computing environment:* available processors, devices accessible for user input and display, network capacity, connectivity, and costs of computing
- *User environment:* location, identity, activity and state of people
- *Physical environment:* lighting and noise level.

All the above-mentioned characteristics imply a change in today's UI's concept. UIs today are *static*, meaning that everyone that uses a software application sees the same interface. Each person using the application must conform themselves to think and work the way the programmers who wrote the interface expected them to act. The interface customization is limited to things like setting colors and fonts, or hiding toolbars. Mobile computing is introducing the need for more *dynamic* UIs according to user profile (user's spoken language, organization, role, experience level) and the profile of the network device (e.g., PC, cell phone, PDA) being used. This tendency leads to the technology of *dynamic, plastic user interfaces* which are generated "on-the-fly" and adapt themselves to the user profile, to the features of the network device and to the context of use.

Context-sensitivity or *plasticity* is defined according to Thevenin *et al.* in [29] as "the ability of a UI to mould itself to a range of computational devices and environments, both statically and/or dynamically, whether it be automatically or with human intervention", by executing reconfiguration of their presentation and dialog (e.g. widget resizing, reduction of a full widget to its scrollable version, graceful degradation of a sophisticated widget into a moderate one, or redistribution of widgets across windows).

Context-aware computing as mentioned by Dey and Abowd in [11] refers to the "ability of computing devices to detect and sense, interpret and respond to aspects of a user's local environment and the computing devices themselves". Ideally the mobile application should have the capacity to withstand variations of the physical and software platforms and the physical environment where the interaction takes place (context of use) while preserving usability. Finally, an application may sense its software and hardware environment to detect, for example, the capabilities of nearby resources.

Context-aware applications dynamically change or adapt their behavior at changes of the context that might occur at runtime, without explicit user intervention. Context-awareness allows an application's behavior to be customized to the user's current situation. According to Dey and Abowd in [11] the application is context-aware if it uses the context to provide relevant information and/or services to the user, according to the user's task.

1.2. Mobile and Wireless Devices: Characteristics and Constraints

We present in this chapter the heterogeneous processing and networking aspects of the mobile computing environment. We describe the characteristics and constraints of the mobile devices and their impact on the mobile applications development. Wireless and mobile devices are small narrowband computing devices primarily characterized by four constraints:

- **Display size** - limited screen size and resolution. The cellular phone may only have a few lines of textual display, each line containing 8-12 characters and limited number of display colors.
- **Limited interaction capabilities** - a limited, or special-purpose input device. A phone typically has a numeric keypad and a few additional function-specific keys. A more sophisticated device may have software-programmable buttons, but may not have a mouse or other pointing device.
- **Limited computational resources** - limited CPU and memory, often limited by power constraints.
- **Narrowband network connectivity** - limited bandwidth and high latency. Devices with 300-baud network connections and 5-10 second round-trip latency are not uncommon.

From the wide variety of wireless devices we included in our study the most commonly used ones, such as the cellular phones and the PDAs. They are assigned an IP-type address and are "always on", acting as part of Internet, sending and receiving information using standard protocols. Devices in new categories are expected to emerge in the near future, such as 3D-immersive and natural language environments.

Mobile phone's display size is between 2 and 10 lines of text and has limited navigation. The images may be of poor quality and it is difficult to enter text. The WAP server connection provides Internet access for the WAP phones. The phone applications should be entirely service-oriented, providing quickly access to information. The PDA is a device with a broader range of capabilities with a display resolution higher than the mobile phones and may support a pointing device or handwriting recognition. The PDA can perform HotSync synchronization over wireless and wired connections with either desktop PCs or network servers and allow loading of small executable code segments.

Mass-market, hand-held wireless devices present a *more constrained computing environment* compared to desktop computers having less powerful CPUs, less memory, restricted power consumption, smaller displays and different input devices (e.g., a phone keypad, pen-based). These wireless devices are interconnected among themselves and to the Internet.

Also the *wireless networks* also have much less capacity, i.e. less bandwidth, higher latency, less connection stability and less predictable availability compared to the wired networks that are presumed for the Internet. Mobile connectivity is highly variable in performance and reliability. This is one of the reasons for which the integration of the wireless telecommunication and Internet technologies is currently done in ad-hoc, inefficient, functionally limited and in not a very less user-friendly manner. The second reason is that the wireless networks and the Internet are intended for two completely different use situations, where different metaphors are used.

All the constraints summarized in this section complicate the design of mobile information systems and require us to rethink the traditional approaches to information accessibility and usability.

1.3. Multi-Platform, Multi-Devices and Multi-User Interfaces

The problem of multi-platform UI development is arising due to the emergence of a variety of devices and channels. With the convergence of the Internet, mobile telephony, and hand-held technologies, a Web application can offer multiple user interfaces, which allows users to interact with the same application running on different devices and computers including traditional desktops, laptops, PDAs and mobile phones. As the types of target devices have exploded, the modern information and communication systems have to present their application logic on different devices with different capabilities.

While in the traditional interface design for desktop, there is a strong standardization among the devices which almost all development tools support, for the narrowly devices there are no such standards not even within devices of the same class. Different kind of PDAs may have quite different display characteristics and input methods. As from HTML browsers to WAP-enabled devices, each target platform, with its distinctly different capabilities, imposes its own constraints to the design of the web site, the most frequently current practice is the development of a solution with a unique UI for each type of device.

Therefore different versions of a web site of the same application for each one of these devices needs to be maintained, such that a lot of time and money consuming work will be done to re-implement a UI again and again, for each platform and usage case. Also the introduction of a new device requires a re-implementation of the UI.

The effort to keep all interfaces consistent increases, as developers need to maintain multiple source code families in order to deploy multiple user interfaces of the same information system on multiple devices. More than that, as the UIs are implemented with an increasing number of software and hardware technologies, developers must learn multiple implementation languages that are evolving over time. In Figures 1-1, 1-2 and 1-3 we show three different interfaces to the same Internet financial management system, for a desktop PC, handheld PC, and cellular phone, respectively.

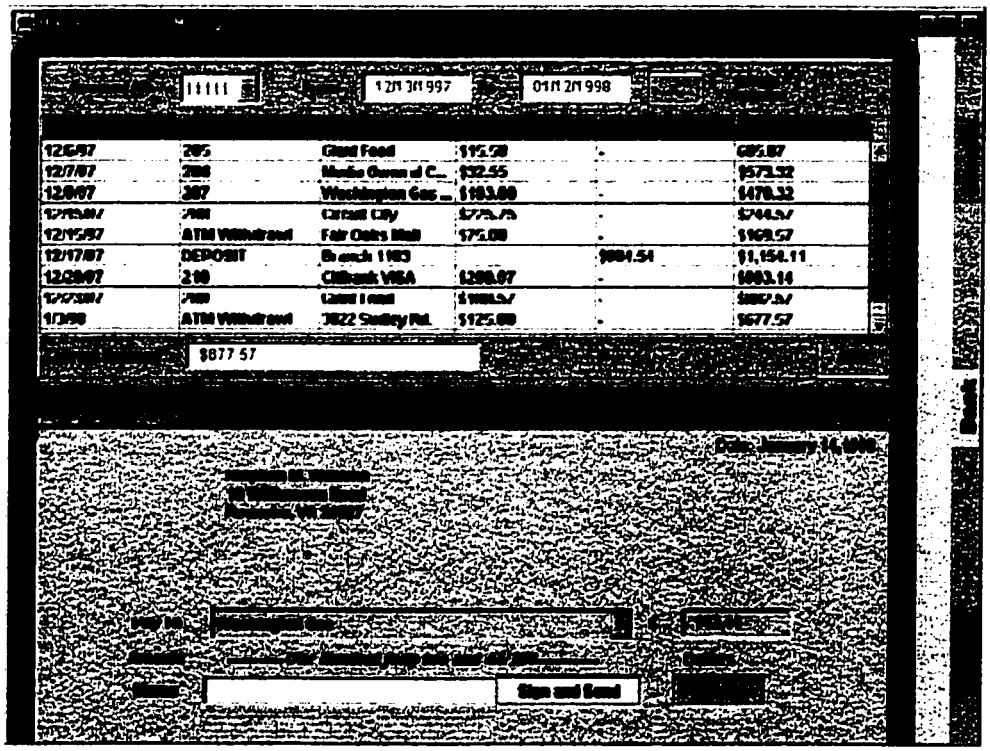

Figure 1-1 a financial application rendered for a desktop PC [11]

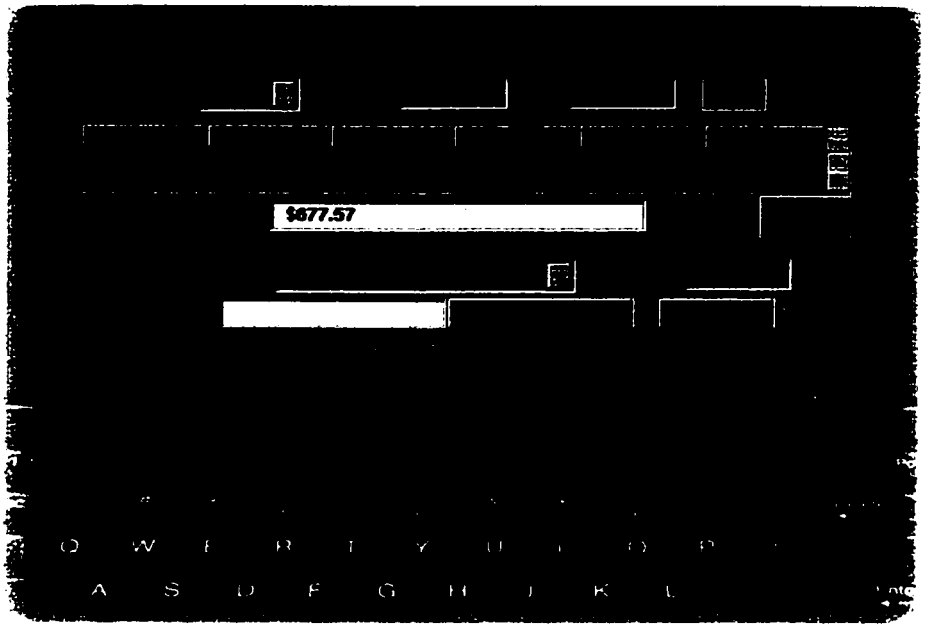

Figure 1-2 Handheld PC rendering [11]

Figure 1-3 Cell phone rendering [11]

As we see in the figures above, the interface content of these UIs differ dramatically in complexity (e.g., PC versus cell phone interfaces).

A solution that can contribute to achieving consistency across several platforms and also greatly reduce the amount of code that needs to be maintained is to build interfaces with some kind of universal language, free of assumptions about devices and interface technology. To develop user interfaces that are portable across devices and operating systems, they need to be described in an abstract manner. This could be achieved through a universal, device-independent markup language that could manage family of interfaces of varying complexity by describing the UIs in a highly device-independent manner, and mapping the description to other markup languages or programming languages via style sheets. The use of a device-independent UI language would allow reusing the interface description that is device-independent with new technologies, reducing the risk of adopting new technologies. This is a guiding principle for UIML language (e.g. the 3 interfaces presented in the figures above are generated from a single UIML description with three style sheets) [11].

Multi-Platform development could be seen from two points of view. The users may move between different platforms while carrying out a single task. Additional, users may want to collaborate on a task while using heterogeneous platforms. Therefor the UIs should be built

in such a way that the users are allowed to perform the same kinds of tasks on devices with different capabilities. In this case, what has to be changed is the set of interaction and presentation techniques to support information access while taking into account the resources available in the device considered. In the figures above we have an example of the same tasks supported by a cellular WAP-enabled phone and a web desktop interface. There is different content differently displayed in the 3 interfaces. The second point of view is to consider different devices with regard to the choice of the tasks to support. For example, phones are more likely to be used for quick access to limited information, whereas desktop systems better support browsing through large amount of information.

We will see in the next section how concepts behind model-based UI development techniques have to be utilized for generating multi-platform UIs for the mobile applications.

1.4. Research Objectives and Methodology

In this research, we are investigating the development techniques for mobile computing environment with emphasis on the user interface component of Internet-based interactive applications. The Web applications for narrowly focused small devices cannot support the full range of the traditional desktop Web functionality due to physical and computational constraints. We will show how the increased connectivity of devices has a profound effect on the UI development tools and methods.

We present the development impacts that are emerging from the necessity of building interfaces for different mobile devices and platforms. We will discuss the required design techniques that can satisfy the ultimate constrained mobile environment while preserving the applications usability. We also justify the need for improving these techniques, for new ways of interface modeling, and for evaluating the usability of mobile interactive systems.

We concentrate in our work on the following questions:

- How can the high level of interactivity of the office computer be maintained or adapted for reduced-capacity terminals such as mobile telephone and PDA without keyboard?

- Weather we should strive for uniformity in the services offered, dialogues and presentation, or should we specialize the interfaces and services to reflect the constraints of each device and/or the context of use?
- How the UI can be adapted to the diversity of devices that exist today?
- What kind of methods is needed for designing for mobile users, their tasks, interaction styles and contexts?
- Why the design techniques and existing forms of modeling software, systems and interaction are or are not appropriate to address the problems of diversity, inconsistency, and accessibility?
- Is the content markup languages are adequate for device-independent authoring?

To answer such questions, a comparative study has been conducted on the development frameworks and techniques for the mobile interactive Internet applications development. Some of the selected frameworks are available on the market while others still under development. Some of the frameworks address the very important issue of applying model-based techniques to the development of mobile UIs. We will expose their strengths and weaknesses, highlighting the areas for improvements. The comparison is performed based on criteria regarding the data processing, data interactivity, security support, the cross-platform or device specific availability, scalability, extensibility, learnability, UI modeling they might involve, UI deployment, delivery mechanism, any contribution to or involvement into model-based methodology and the use of the model-based methodology. We will compare the information presentation techniques and interaction mechanisms for supporting the user interaction mainly with the mobile phone and PDAs as it is an important concern to overcome the limitations (layout features, screen sizes and resolution, interaction capabilities), imposed by the different platforms.

We also discuss the guidelines that can help developers to select one of the frameworks, for a particular type of application and context of use. Our goal is to understand the wireless development, why one development approach is suitable for a particular type of application on a particular device, and ultimately to be able to make strategic decisions for building highly effective mobile Internet solutions based on design and usability guidelines.

2. CHAPTER II – Usability Challenges for Mobile / Wireless Interactive Applications

2.1. Wireless Users

The mobile user is anyone using a wireless device to access an Internet-based interactive system. We should envision that the mobile users:

- Will use the device or/and the application as a convenience, thus more frequently, for less time in short session and with less well-thought goals or plans;
- Need simplicity, and wearability rather than superfluous functionality;
- Will have small, specific tasks that need to be accomplished quickly.

Wireless users are much more likely to use their devices in public and while mobile. This makes their experience short-lived, more open to distraction, and less open to pure browsing. Wireless subscribers have a different set of essential desires and needs than desktop or even laptop Internet users. Although people will not necessarily expect these devices to act like "regular" computers, they expect similar capabilities and accessibility (anywhere, anytime, any device) on a small screen device as on a big screen desktop. Today, mobile users expect handhelds to help them manage information by being able to select data, drill down for more detail, analyze it, input new data and perform transactions. Instead of trying to squeeze desktop enterprise applications onto handheld, developers should build simple solutions specifically for mobile users to give them access to information. There are an increase number of user categories, each requiring a separate interface.

The assumptions the PC interface designers make, regarding the user of their applications situated in a comfortable, free of distractions environment, with high concentration directed towards performing tasks and achieving goals, having suitable access to input devices and easily submitting required or requested information, do not accurately describe the mobile user. These users are often situated in heterogeneous environments, where they are distracted by the multiple tasks they are performing simultaneously. These users tend to avoid using functions that require extensive interaction.

Whereas in the traditional office desktop application there are some factors the user has a chance to configure to his convenience, so he can better perform the tasks and the overall application usability is increased, this is not possible on the narrowly devices. On these devices the tendency is to miniaturize the applications and the user is not really allowed to configure any display or application parameters.

2.2. WAP Usability Findings from Nielsen Survey

During the fall of 2000 Nielsen Group [6] performed a usability evaluation for the WAP services. Tests were conducted mainly from the perspectives of the efficiency in task execution with the final objective of developing a more usable mobile Internet strategy. In this section we will present some results from Nielsen's survey. We used these results to derive the major usability requirements not only for WAP phones but for any other small devices with some kind of Internet access.

Some of the usability issues for the mobile/wireless Internet applications are related to how the options are described to the user (labeling), to the availability of search facilities and bookmarking, to the information that users managed to download, while the others come from the users' expectations of WAP in light of their assumptions about the PC Internet counterpart.

According to Nielsen, the usability of current WAP services is severely reduced because of a misguided use of design principles from traditional Web and GUI design. On a small-screen device, designers must *conserve screen space* by showing only the most important information. Given the cumbersome and slow process for moving between WAP screens, it is even more important for WAP design than for other small-screen designs to optimize the communicative value of each screen. The traditional websites design guideline of repeating the user's choice on the destination page for confirmation cannot be applied to WAP design. The users often faced unclear labels and menu choices written in special language "invented" by the WAP designer, while the users want standard terms for standard features. The need for simple language is strong in WAP design, because there is no room to explain non-standard

terminology with rollover effects, icons, or captions.

Several WAP services that were tested were unnecessarily hard to use because of a mismatch between their information architecture and the users' tasks. The traditional Web also suffers from poor task analysis, with many sites structured according to how company management thinks rather than how users typically approach their tasks. Although poor task support is a serious usability problem for a big-screen website, it is a usability catastrophe for a small-screen WAP service. Therefore a *very precise task analysis* will be necessary for WAP services to succeed. With the big screen, users can see many more alternative options, and thus it is not so critical as for the small screen that designers pick exactly the right ones at each step. *Mobile applications need intelligent interfaces to assist users in performing their tasks.*

Another WAP usability finding from the Nielson survey, which has not been seen on the Web, was a lack of clear differentiation between the services. Nielson conclude that the differentiation cannot come through graphic design because of these small, mainly-text screens; it will come from a better writing style optimized for very, very short and useful content. This has to be the real way to distinguish WAP services. Mobile services must target users with immediate, context-directed content.

Sometimes users failed in accomplishing their tasks because they did not scroll the WAP screens to see content and menu options that were not visible on the first full screen. Scrolling is less common in WAP navigation than in PC-based Web navigation. The usability survey found a lot of screens with poor typography and for the WAP users who do not want to read a lot of text it is painful to have to scroll through page after page of small snippets of text and try to piece them together in mind.

One usability issue that should be addressed in generation of WAP interfaces, according to Nielson is the "Back" feature for immediate redisplay of the user's previous location. It is unacceptable to expose users to further connection delays as they retrace their steps back through an unsuccessful navigation path. The phone must cache many previous page views

and render them immediately. Also, when backtracking, users should be taken to their previous location on each page. Although the users accept that they might find themselves on the wrong track and have to go back a few steps to choose a different path, all too frequently, they faced dead ends or error messages that blocked their path. Although instinctively most of the users looked for and found a "Back" button, they were not always able to retrace their steps. Sometimes they had to wait to be connected just to go back one step; other times they were simply unable to go back any further or were forced to go back to the very beginning. This wasted a lot of users' time and money.

It may be a consequence of the limited real estate on a WAP screen, but even when a search facility was available, its location was not clearly signaled on the networks' portals. The result was that users were frequently unable to find sites that might have given them the information they were looking for.

Bookmarks are critical to users efficient use of WAP. However, users were greatly frustrated that the Nokia 7110e can only store 10 selections at any one time. The Ericsson R320s stores 25, but even this number is too small to accommodate the number of selections a user is likely to need.

The users found a lot of unhelpful or annoying messages like "Please try again" or "Internal server error" which doesn't tell whether it's a problem of what the user has done on his phone or whether it's a problem with the site that the user is trying to access. Consequently they wanted some more detailed and user – friendly message or maybe more of an explanation as to why a service is not available owing to "too many people dialing at the same time" or "it's your connection problem".

Navigation on WAP-enabled phones is based on the same principles as the Internet and its success is critical to the efficiency of the system itself. Menus should hold a small number of options and would be nice that users can customize their own menu so that their favorite sites or interests come up automatically. Generally speaking, WAP users' navigational aids are restricted to menu options, search engines, and bookmarks. However, even these basic

functions did not run as smoothly as the users expected at the time these tests were conducted.

We need to *simplify navigation*, to better direct the users to products and services, to *simplify the search functionality* and to allow scaled images in the graphical layout. The consolidation of the search function is needed, so that users would only search that portion of the database related to their current navigation with results in the simplification and contextualization of key features.

Once users succeeded in finding a site that interested them, the next hurdle was to read the text from such a small screen. The small screens would not be a problem for the user if they presented clear screen content, *concise headlines*, *brief extracts* or short openings to sum up properly everything that lies beyond. For news one- or two-line headlines that can be transmitted quickly and understood at a glance would be perfect for the instant work mode of mobile users. As most of the mobile users, most of the time, are online for a handful of minutes, they should be given a *personalized summary* of important information and be allowed to *easily drill down* for more detailed information. If the item is still intriguing, they can download the full text of the story. Certainly, these facilities must be offered with respect to the small screen difficulties in scrolling or scanning.

Another finding from the Nielsen's survey is that sometimes the phone is waiting for the user to input more information, but doesn't make this clear to the user. Often, the reason is that the phone is waiting for the user to enter information into text boxes that can only be found by scrolling further down the page. And sometimes, there is no indication that there is any more text to scroll down to. The selection lists should be used to minimize the potential for error occurring when the user is asked to type in too much information. Another useful, if not vital, facility is the ever-present availability of a "Help" button. However, help functions are sometimes offered with less than reassuring consequences.

Wireless connections get dropped a lot more than regular Internet connections. One of the greatest problem users had with WAP is the inability to connect to services because of a

variety of failure modes—networks were down, the phones crashed, or the service itself was down. Users don't know and can't find out because sometimes when all they get is an incomprehensible error message. This has to be avoided through a good design. Connection failures are particularly annoying for WAP users since they are paying for airtime. Connectivity and the cost to the user of being online are critical to the future WAP development. Therefore we should focus on tasks that could be completed in a short amount of time with as few screens as possible.

Nielson conclusion was that WAP performed poorly in terms of learnability and efficiency. Often when the system was used, errors rather than being "few and far between," plagued the system. In terms of being pleasant to use, WAP generally failed to satisfy. The users accepted pretty well the screen's limitations and realized that there has to be a trade off between portability and ease of use. Some findings revealed the need for more highly interactive, supportive, and easy-to-use user interfaces and for determining what aspects of interface design are most important to be factored out.

Given the nature of the mobile, wireless user experience, we have to be extremely discerning about what we want to present to the user, and how to present it, when developing the wireless applications. It is extremely important to use the available "real estate" on the display efficiently. Only menus and navigation aids related to the immediate task should be visible; everything else should be hidden until needed. The lesson learnt from the Nielsen's survey is that when designing UIs for mobile devices, especially for heterogeneous environments, we have to consider not only the capabilities of the involved device, but also the special requirements demanded by their target users - the mobile, wireless subscribers.

2.3. Mobile Computing Guidelines

Handheld devices have their own user interface requirements in terms of space and memory. The user interface requirements for handheld devices are different from those for desktop computers. For these reasons, the user interface programming guidelines for applications running on desktop computers and hand-held devices are different, as we will see.

In order for the PDA to function as a hardware extension to the user, enhancing abilities such as recall, calculation, and mental organization, efficiency and ease of use are primordial for mobile applications running on PDAs. In order to achieve this, the developers should use the following guidelines extracted from [4]:

- **Create fast applications**

  PDA users want instant access to information. The total time needed to navigate, select, and execute commands can have a big impact on overall efficiency. To maximize performance when working with the PDA, the UI should minimize navigation between windows and opening of dialog boxes.

- **Match use frequency with accessibility**

  PC user interfaces are typically designed to display commands as if they were used equally, when in reality, some commands are used very frequently while most are used only rarely. Similarly, some settings are more likely to be used than others. More frequently used commands and settings should be easier to find and faster to execute.

- **Create easy to use applications**

  Advanced commands should be easily accessible but should not be in the way.

The following guidelines allow for making the application's UI intuitive, easy to use and consistent with other applications on the device such that users work with familiar patterns.

- **Data Entry Guidelines:**

  For the Palm device to work efficiently and comfortably with the displayed information the users need to have a variety of input modes.

  - Platforms that support different input modes for the user to enter data: infrared, note pad, touch screen, onscreen keyboard (buttons, check boxes, and popup lists provide a quick way to enter settings and select options), Graffiti software and foldout or clip-on full-size keyboards, HotSync (the user can type data on the PC and download it to the Palm OS device), should be chosen by the developers;

- Users should be allowed to perform basic data entry in place;
- The cursor should be ready and visible if there's only one field for text entry;
- Details dialog should be provided for more elaborate data entry;
- Dialog boxes should not be nested too deeply;
- A graphical element should provide just one application's functionality. The user should interact with the application through either a button, menu, or popup list;
- Whenever a field for user input is available, we have to ensure that: System keyboard is available via shortcut or via menu, Graffiti input is possible (regular strokes and shortcuts), Cut, Copy, Paste, and Undo are possible.

- **Command Execution Guidelines:**
  - Users should be able to execute commands by using Command buttons or Menus, Graffiti menu command shortcuts, buttons on command toolbar;
  - Menu shortcuts should be provided whenever possible.

- **Guidelines for Screen Layout:**
  - Simple screen layout
  - In the title bar for each screen, we should provide both the application name and the name of the screen, if possible. Otherwise, we should provide the most relevant information;
  - In order to maximize the screen real estate available to the application, we should not use borders;
  - Resources provided with the development environment and the recommended values for width, height from the Palm OS Programmer's API Reference should be used;
  - Buttons should be aligned with the bottom edge of the screen. For text surrounded by borders, there should be one pixel above and below the font height should. For controls that can be displayed in groups, there should be at least two pixels to the left and right of the text label. The exception is command buttons, which require wider margins to accommodate the rounded border;

- **Guidelines for Dialog Box Layout:**
  - Online help should be provided for dialogs;
  - Labels should be bold and editable items nonbold;
  - In the details dialog, the label should be right-aligned and the editable field should be left-aligned;
  - In dialogs, a space of four pixels should be left between the edge of the dialog and the buttons, and between the adjacent buttons;
  - Dialogs should be aligned with the bottom of the screen. The screen title bar should be left visible if possible.

2.4. Development Challenges and Trends

Mobile computing imposes new challenges in UI design and development, as user interfaces must run on different computing platforms accommodating the capabilities of various devices and the different contexts of use, while preserving consistency and usability.

Challenges are triggered also because of the universal access requirements for a variety of users and devices [10]. All these devices need to be connected while the computers and network services have only limited ability to communicate with each other and often have widely different formats of data processing and interaction. We claim that our existing forms of modeling software, systems and interaction do not adequately address problems of diversity, inconsistency, accessibility and integration. Therefore we need to prepare new ways of modeling, designing and evaluating the usability of mobile interactive systems.

To satisfy user and business needs for "anyone, anywhere," one must design effective information environments not only for mobility but also for contextual use in a situated environment. The challenge that designing for mobility brings, is the need to take into the account the "mobile user" philosophy opposing that which has been employed by traditional software developers to date, and the new heterogeneity of computing platforms, each of them with its own device capabilities.

2.4.1. Design for Device Constraints

The devices and medium constraints play determinant roles in defining the design strategy for the mobile applications. Different devices always posing new constraints and have different interaction capabilities, provoke in the mobile applications the need for reconfiguration of UI beyond the traditional UI change.

Most of the technology developed for the Internet has been designed for desktop PCs with high-resolution screens, graphical displays, ample memory and computing power, supporting many types of user interaction, voice capabilities, the ability to download, store, and run executable code (e.g., a Java-applet, Active-X) while connected through medium to high bandwidth, generally reliable data networks.

Because of the devices' constraints, the narrowly focused platforms cannot support the advanced features offered by classic computing platforms such as desktop applications with rich interfaces that employ widgets such as windows, icons, and menus to interact with the user. While the existing PC Web applications/ pages are graphics-heavy having moving images, featuring particular style fonts or buttons and offering a lot of content to the customer, the wireless applications cannot replicate this look and content. Therefore the UI of a wireless handset is fundamentally different than that of a desktop computer and much more restrictive.

When designing for small devices and mobile computing in order to achieve the same quality of usability as for regular PC devices, we also need to consider that the device's characteristics and constraints have an immediate effect on the task execution time.

The differences between traditional UIs for graphical desktop interfaces and UIs for mobile devices involves the following areas:

- **Functionality and Complexity:** Traditional desktop applications' UIs may have complex functionality. These UIs support a large set of interface elements, in contrast

with mobile UIs that typically are not as complex nor do they support a great deal of functionality.

- **Input Mechanism and devices:** In traditional desktop UIs, the keyboard and the mouse are the primary forms of input devices. In mobile devices, typically the input mechanism is through small keyboards, touch-sensitive screens and styli for PDAs, numeric keypads for mobile phones, a constraint that places restrictions on the types of applications and UIs that can be supported on hand-held devices. Changing from a mouse to a stylus on a touch pad requires different interaction techniques.

- **Screen capability:** Desktop devices have typical screen sizes with resolution up to 1024x1024 pixels. In contrast, mobile devices have limited screen size in the range of 200x150 pixels (PDAs). In particular, mobile phones have very limited screen size of a few square inches. Most hand-held devices support only monochrome displays.

- **Delivery mechanism:** In traditional UIs, the generation of UI can be done either remotely if the application is distributed and delivered to the device, or it can be done on the device itself in the case of a non-distributed application. However, in the case of a hand-held device, the actual UI is created separately and just delivered to the device. This mechanism is also used because of the limitations of wireless networks. However, the generation of the UIs for mobile applications is done remotely in majority of the cases and the delivery typically takes place through a gateway or an intermediate server over a network. The low data rate for wireless devices also places a restriction on the size of the user interface that can be delivered to the device.

The mobile applications needs to accommodate the devices' proliferation and variations such as processing, operating systems, storage, bandwidth, physical and display characteristics. The bandwidth requirements are driven by the amount of content per page, the richness of the media being delivered, the number of pages accessed and the size and frequency of requests being sent. This constraint implies for the applications to present different content sets for

different users and platforms. The use of multiple URLs to distribute comparable content to users will pose significant usability concerns.

The most obvious impact of mobile devices diversity is that developers will now have to create UIs that work with vastly different sizes and characteristics of displays, therefore the current techniques that make implicit assumptions about device characteristics need to be changed. The display resolution along with the reduction of the design space is the mobile computing platform's most difficult constraint to deal with. Because of screen resolution differences, an optimal layout for one display may be simply impossible to render on another device. Reducing the graphic might make it meaningless, while scrolling large graphics both horizontally and vertically is an unwieldy, unfriendly approach.

In order to support the above-mentioned new form of user interactions (note pad, touch screen, onscreen keyboard) and to overcome the display limitations for these small devices, new presentation and interaction mechanisms for mobile computing and communication devices are needed. We should carefully select the presentation techniques that convey information quickly. The "deck of card" metaphor used by WAP [8] and the "screen" metaphor related to MIDP [17, 18] both allow the UI information to be presented on a cell phone's small display and be manipulated through keypad solutions. They will be presented in Chapter III.

The following design considerations will have a determinant role in achieving the usability requirements as presented in section 2.2 for both cellular phones and PDAs devices, while taking the devices constraints into consideration:

- **Feature Restriction**

  Many of the features common to PC applications are fundamentally not usable on a PDA or cellular phone. A PDA can function ideally for rapid data entry, rapid information retrieval, automated information collection, and higher-level manipulation. In a user centered design model, the application architects should prioritize user goals according to these feature categories. Feature restriction may then be applied to lower priority items.

- **Information Filtering**

  Typically a wireless user will require preprocessed or summarized information. A PDA user is generally distracted by his environment, which makes absorbing larger quantities of information difficult. It is therefore essential to build an interface layout that presents important information in a form that is clear, uncluttered, and easily identifiable. In order to comply with this criterion, it is suggested that information be classified according to frequency of retrieval, and level of required detail. The most utilized information should be immediately accessible. When more detail is required, the user should have the ability to access suitable views by means of additional navigation. Essentially, retrieval frequency and navigation complexity should be directly correlated. A possible alternative is to have modes that show local-only information (things that can be seen now) and more global information. How and when the user would switch between these modes will determine the ultimate utility of the device and the service.

- **Interface Compression**

  As we saw the physical size constraints of a PDA or cellular phone dictate a certain level of interface compression. Still the interface needs to be visually appealing. Text must be of a smaller font, GUI controls must be resized, and icons modified. Although feature restriction and information filtering somewhat alleviate this problem by reducing the number of objects that are displayed on screen at any particular time, interface compression is a fundamental issue that must be addressed. Particularly two aspects require our attention:

  - **Object Recognition**

    Modern GUIs are composed of controls, icons, and other objects whose functions are hinted at visually. However, interface compression can greatly distort their identification. Past a certain size threshold, many visual cognitive cues can no longer be recognized. As a result, careful graphical design is essential. In the case that an object cannot effectively be miniaturized without compromising ease of use, new cognitive icons should be designed.

- **Object Manipulation/Navigation**

  Small controls are difficult to select. Cluttered input areas are error prone. GUI developers must consider these facts when designing the interface layout. An efficient interface must be forgiving of user error and variation.

2.4.2. Design for Multi-Platform

An approach for multi-platform development for existent PC Web applications would be to *scale* the current UIs to radically different platforms. This implies that the UI would need to be *retargeted* across different browsers and hardware platforms [10]. The adaptation of the content displayed on PC to the small wireless devices screen should consider the wireless browsing model, which is different than the traditional one. We present in Chapter III a proposed system for automatically transformation and adaptation of a PC Web on any device.

Another approach for the multi-platform development is given by the *model-based techniques* for generating UIs [5, 6]. Model-based user interface development tools use different kinds of high-level specification of the tasks that users need to perform, data models that capture the structure and relationships of the information that applications manipulate, specifications of the presentation and dialogue, user models etc, and automatically generate some parts or the complete UI. The UI specification is automatically translated into an executable program, or interpreted at run-time to generate the appropriate interface. The aim is the portability across multiple types of devices.

The currently existing model-based systems use heuristic rules to automatically select interactive components, layouts, and other details of the interface, such that the connection between specification and final result can be quite difficult to understand and control. There are significant limitations on the kinds of interfaces they can produce, and the generated UIs do not have the quality of the UIs that could be created with conventional programming techniques.

Model-based UI systems take an abstract model of the user-interface and apply design rules and data about the application to generate an instance of the UI. Transformations from an abstract model are used instead of transforming widget by widget because a model can have more semantic information about the interface. For example, a model could determine the relationships among widgets in a GUI dialog box by their layout, and keep those relationships in a transformed interface for a PDA or a cell phone. Transforming only at the widget level would lose those relationships.

Declarative model-based techniques involve user interface modeling techniques that require the user-interface to be described from a position of abstraction. The *UI model* expressed by a modeling language should provide a formal, implementation-neutral description of the UI. The language should be declarative, so it can be edited by hand, but it should be formal so that it can be understood and analyzed by a software system.

Currently the model-based techniques using mainly the *task and domain models* do not generate quality interfaces. One more limitation of some of the earlier systems was the lack of user control over the process of UI generation. Having more user-control over the UI development process and having more usable models could rectify some of the deficiencies of the model-based approaches.

The *challenge* for the *UI modeling* is the creation of *knowledge bases* that describe other various components of the user-interface, besides the task, the domain, the presentation, the dialog, such as the platform, and the context. These knowledge bases can be further exploited to automatically or semi-automatically produce a usable UI matching the requirements of each context of use. With current model-based techniques the task analysis is performed for obtaining a single UI that is adapted for a single context of use. As with the explosion of computing platforms, and contextual conditions, the UIs need to adapt to multiple configurations of the context of use, there is a special need for modeling tasks that can be supported in multiple contexts of use, considering multiple combinations of the contextual conditions. Therefore the model-based techniques should help designers to recognize and accommodate the unique contexts in which mobile computing occurs. It seems critical that

models should be empowered with some elegant way to segment different aspects of UI design that are relevant to different contexts of use and to isolate context-generic issues from context-specific ones.

We will show in Chapter IV how the model-based techniques range from relatively low-level implementation solutions, such as the use of abstract and concrete interactor objects, to high-level task-based optimization of the interface's presentation structure.

As currently there is a lack of support for the observation of *usability guidelines* in the development of mobile user-interfaces, the model-based UI development environments need to incorporate usability guidelines and to provide support for applying these guidelines. We believe that user interface modeling will be an essential component of any effective long-term approach to developing UIs for mobile computing.

2.4.3. Design for Context-Awareness

The challenge that designing for contextual use in a situated environment brings is that a user's attention can be divided depending on the location of device use and that the device itself can both facilitate and frustrate the user's goals.

As mobile computing increases the probability of environmental change while the user is carrying out the task, the challenge for the UI design is to continue to support users in accomplishing their tasks while the context evolves in time, space, and resources. The designer needs to match the UI configuration (e.g., look and feel) with the set of constraints imposed by the new characteristics of the resulting context of use. A context sensitive system would choose an appropriate representation for the UI based on the user tasks or activities and environment, when changes are of sufficient degree to justify a change in the information representation. The incoming context should be used to determine the particular actions that need to be performed in the new user situation. Context-aware applications need to be able to display context information, capture it for later access and provide context-based retrieval of stored information [33]. Also the context-aware applications need to:

- Acquire context from unconventional sensors: outdoor GPS receivers, indoor positioning systems, Active Badges devices, video image processing or floor-embedded presence sensors.
- Acquire context from multiple distributed and heterogeneous sources
- Detect changes in environment in real-time and adapt to constant changes.

In summary, mobile computing requires that UIs be sensitive to:

- *Platform*: by having presentations that adapt to screen surface, color depth, screen resolution and dialogs that adapt to network bandwidth;
- *Interaction*: by having mechanisms that remember previously used interaction techniques, windows sizes and locations, and respect user preferences for these presentations;
- *User*: by adapting to user experience level, system and task experiences, skills, conventions, and preferences.

One of the adaptation techniques is the *interface tailoring* [2] for customization and optimization of an interface according to the context in which it is used. Interfaces can be tailored to tasks that different segments of the user population need to perform most often, to the level of use and experience of users, to the physical abilities of users, to platform characteristics, etc. Tailoring involves modifying the interface design. The simplest level of tailoring happens at the concrete level of an interface specification where features such as the layout, colors and fonts of an interface are changed. More sophisticated tailoring can happen at the abstract interface specification where the dialogue is modified, for example to shortcut certain steps, to rearrange the order for performing steps, etc. At the highest level, new tasks might be defined by composing existing tasks.

Another aspect of UI adaptation to the context is the "*location-awareness*". One of the unique aspect of mobile devices is that if combined with GPS (Global Positioning System) technology they could become aware of their position, providing the opportunities for location-aware applications where the information presented to the user is strongly linked to the location where the device is being used. The degree to which the mobile application is

coupled with the location of the devices and how this location is made available to users is a key design decision in supporting different interaction styles. The Lancaster GUIDE system [30] is an attempt in exploiting the "location-awareness".

For mobile applications the *variations* in the mobile *systems infrastructure* may dramatically effect interaction and it is essential that interaction styles and interfaces provide access to the information reflecting the state of the infrastructure. The UI must be designed to cope with the level of uncertainty that is inevitably introduced into any system that uses wireless communications. A key design element of the UI is to support the infrastructure variation, how to present to the user the information obtained from the platform.

On the other hand the existing software infrastructure needs to be improved in order to provide better support to adaptive mobile applications. These applications need to determine how best to adapt, while preserving the ability of the system to monitor resources and enforce allocation decisions. By using local resources to reduce communication and to cope with uncertainty, adaptation insulates users from the vagaries of mobile environments. The client-server model should be extended as shown in Figure 2-1 for the reasons that we further present.

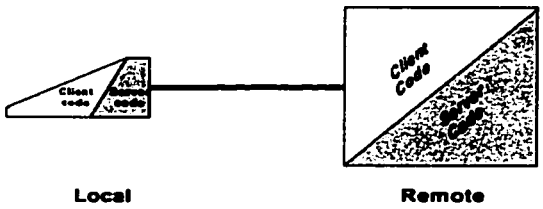

Figure 2-1 Extended client-server model [1]

The resource limitations of clients may require certain operations normally performed on clients to sometimes be performed on resource-rich servers. On the other hand the need to cope with uncertain connectivity requires clients to sometimes emulate the functions of a server. As the circumstances of a mobile client change, it must react and dynamically reassign the responsibilities of client and server. Mobile clients must be adaptive and sense changes in its environment, make inferences about the cause of these changes, and then react appropriately. These imply the ability to make global estimates based on local observations.

To detect changes, the client must rely on local observations. For example, it can measure quantities such as local signal strength, packet rate, average round-trip times, and dispersion in round-trip times. A change in a given quantity can be due to a multiplicity of non-local phenomena. For example, packet rate will drop due to an overload on a distant server. But it will also drop when there is congestion on an intermediate network segment. If an incorrect cause is inferred from an observation, the adaptation performed by the client may be ineffective or counterproductive.

We end this section with the extremely important conclusion of the adaptation being the key to mobility.

2.4.4. Low-level Frameworks versus High-level Frameworks

We consider as low-level or high-level framework the existing and under development techniques for mobile UI development. We will describe in this section the reasoning behind the split. Under the umbrella of low-level frameworks we consider the wide variety of existing wireless Internet technologies and platforms such as: Web Clipping [7], WAP [8], XML [9], UIML [11], and J2ME [13]. The low-level frameworks supports markup languages for handheld and wireless like HTML used with Web Clipping, WML used with WAP, XML, UIML, scripting languages such as WMLScript and programming languages like Java. They all address the fact that screen size, input capability and bandwidth are limited. The low-level frameworks that we consider are:

- **Device-specific**: Web clipping for PDAs and WAP for cellular phones and support the development of unique UIs for a specific type of wireless device;
- **Cross-platform** like XML, UIML, J2ME with Java and strive for consistency between the interfaces on a variety of platforms. XML along with XSL are among the firsts attempts to abstract the UI functionality from its look and feel.

The *device-specific low-level frameworks* take into the consideration the unique constraints posed by each platform, displaying information and interacting with the user in specific ways

for each device. The limited screen size, the lack of a mouse and the other new form of user interactions mentioned in sections above, require a different user interface metaphor than the traditional, standard desktop GUI model, such as the "deck of cards" metaphor implemented by WAP. Web clipping is an HTML-based technology for building applications optimized for handheld that runs on Web servers. Web clipping architecture is included in the Palm OS platform. WAP specifies an open, standard architecture and network protocols for wireless Internet access, mainly for the cellular phones and pagers.

We consider the *high-level frameworks* (see Chapter IV) the development environments, still under current research, based exclusively on UI model-based techniques. The aim of the high-level frameworks is to allow cross-platform development of UIs while ensuring consistency not only between the interfaces on a variety of platforms but also in a variety of contexts, and maintaining the usability. They mainly address the issues of multi-platforms, multi-devices and multiple UIs.

The high-level frameworks make use of abstract, platform-neutral models to describe the user-interface, which we claim greatly facilitates the development of consistent, usable multi-platform user-interfaces for mobile devices.

The high-level frameworks provide support for a new collection of constraints that are no longer imposed by the computing platforms themselves, but rather by other factors, such as the type of user, the external resources manipulated by the UI (e.g., the network, the ambient environment). In addition, they will help designers to recognize and accommodate the unique contexts in which mobile computing occurs.

One of the high-level frameworks issues is that the human computer interaction involved in mobile systems extend well beyond the interface provided by the device and have significant impacts on the infrastructure. The high-level frameworks should allow UI reconfigurations depending on variations of the context of use, through the model-based approach.

High-level frameworks allow to a certain extent the integration of the low-level frameworks for delivering the best possible, optimal wireless solutions. The concept of device independent design and generation of user interfaces is actually based on the XML-technology. In Chapter IV we will describe attempts that have been made to integrate task and object knowledge into the UI development process and its underlying representations using the XML technology. Also we will see the strategies used by the high-level frameworks to cover the problems raised by context-sensitivity.

2.4.5. Proposed "Universal" High-Level Architecture for Developing Mobile Applications

Our proposed framework would try to integrate the foundation technologies for both the low-level and high-level frameworks to leverage a more interactive and dynamic wireless application environment (see Figure 2-2). The "universal" architecture for development of the interactive mobile applications will allow the flexibility of developing context-sensitive UIs for multiple devices and multiple platforms, while respecting the unique constraints posed by each platform and ensuring the usability.

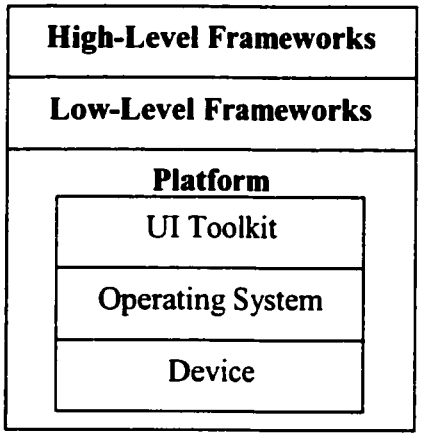

Figure 2-2 Universal Architecture for Developing Mobile Applications

- **Platform** is a combination of a device, operating system and toolkit. There are the following OS for wireless development: Palm OS for Palm devices, EPOC (Symbian) used by Motorola, Nokia, Ericsson, Windows CE a scaled version of Windows 98 used

by Pocket PC (can support current MS desktop applications), Java MicroEdition designed to run on limited memory devices. It seems clear that the facilities that are provided by the "operating system" will continue to expand. Some of the capabilities must be provided at a very low level. For instance, access to information about the input, output and communication capabilities of devices must be provided to the application software, so it can make intelligent choices about the user interface. For instance, Windows CE for palm-size devices seems to make it impossible to find out what kinds of hardware buttons are available on the PDA. Another example is that none of today's networking interfaces makes it possible for an application to decide how fast a connection is available. In fact, it is usually impossible to find out if you are connected at all, so applications freeze up for minutes waiting to see if the network might respond, when the operating system could easily tell that the machine is disconnected. This is unacceptable. Interfaces of the future will need to make intelligent choices based on knowledge about the current capabilities of the device and its connections to the environment.

Other capabilities might be provided on top of the OS, but alternatively might be part of it. For applications to run on a variety of devices, a portable infrastructure must be provided, possibly like the Java run-time library. There will also need to be high-level protocols for accessing information on distributed files. End-user programming support will be needed across multiple applications. Ideally, all of these required capabilities will be available, but not bundled with the operating system. Platforms are needed that allows applications to be built once and run on multiple networks and handheld.

- **UI Toolkit** normally is the software library upon which an application's UI runs (JavaAWT, JavaSwing, MFC) but, in the context of this thesis, as newly developed markup languages like WML, HTML, UIML are capable of representing user interfaces, we consider them "toolkits" as well. Toolkits typically provide both a library of interactive components describing the widgets like menus, buttons and scrolling bars and their behavior, and an architectural framework to manage the operation of interfaces made up of those components. The use of library of reusable components makes UI construction much easier than programming interfaces from scratch. Therefore the

toolkits allow for achieving the goal of consistency between multiple interfaces. The differences in input mechanisms between the desktop computers and the mobile devices lead to the fact that a mobile user interface toolkit for small screen devices should be independent of any particular input mechanism. We claim that there is a need of a general, very highly abstracted user interface toolkit for applications that do not need any platform specific user interface elements.

3. CHAPTER III – Low-Level Frameworks for Mobile Applications

3.1. HTML and Web Clipping Framework

3.1.1. Motivation and Objectives

The HTML 3.2 subset of HTML was developed for small devices and mobile access, taking into consideration their hardware restrictions. The goal is the creation of friendly and small contents in the case of narrow bandwidth and low speed wireless networking, the ability to perform simple and easy operations and the Web accessibility guidelines: prompting, and providing appropriate documentation and help, providing flexible editing views, navigation aids and access to display properties for authors.

The reason behind the Web-clipping technology was to provide a fast new and efficient way for busy people on the move to access information on the web. The goal of Web Clipping is to bring the user just the requested information, without all the links and graphics to slow down the information transfer. It can be used from low-power handheld Palm VII organizer computer with its tiny screen, battery-powered operation, and relatively slow and expensive wireless connection to the Internet.

3.1.2. Features Description

Web Clipping is a server side technology to generate Web pages for PDAs that dynamically transforms an HTML document into a light subset of HTML 3.2 understood and accessible by the PDA (Palm VII or Palm V with wireless modem). It is an HTML-based technology for building applications optimized for handhelds that also runs on Web servers, allowing for rich graphical displays and interactions, and flexible input. Web clipping is the best choice for network-resident applications that require a richer display and more user interaction than typical WAP applications. It's well suited to presenting multiple pieces of information simultaneously, as in online catalogs, travel guides, real estate home finders or online trading applications.

Web clipping *optimizes HTML* for display on small handheld screens and the data transfer for wireless networks by storing static HTML content on the handheld and thus minimizing the amount of data that travels over the wireless link. As with WAP, some kind of network connection must be available as the data access and most of the processing resides on a Web server. In which concern the data transfer, Web clipping applications can send and receive data over a variety of wireless and wireline connections. Web clipping is supported through either built-in or add-on features by every Palm Powered handheld.

The *Web clipping architecture model* [7] includes a Web clipping client-side applications that runs on a Palm handheld, proxy servers and content servers. There are no local database or data processing capabilities. The client-side applications is constructed in HTML and translated into the Web clipping application format, a subset of the HTML 3.2 standard. The proxy servers handle translation between the HTML on the content server and the Web clipping HTML subset that the handheld understands.

The Palm VII organizer has a mini-Web site named Palm Query Application (PQA). A typical PQA is written in HTML, contains a form or a list of hyperlinks that request additional information either locally on the handheld computer or remotely on the Internet. Typically, the users composed simple PQA query (e.g., a request for a stock quote) is sent to the proxy server and from there to the appropriate Internet server.

Figure 3-1 A One-Page PQA [7]

Figure 3-2 A MultiPage PQA [7]

The Web clipping is this HTML results page that answers the query. Clippings are typically small pages generated by a CGI script from the user's query, though they can be static pages on an Internet server as well.

Figure 3-3 Typical Web Clipping Page [7]

The PQA is created and compiled using the Query Application Builder (QAB) on the desktop computer and installed on a Palm VII organizer for testing, or they can be tested directly on the computer using the Palm OS Emulator (POSE) [7]. When tapped, the PQA is rendered by the Web Clipping application, an application otherwise invisible to the user.

Handheld applications are small executables designed for a Palm OS handheld but usually created on a Windows desktop computer. A *Web clipping application* is a set of HTML

pages compressed into a special PQA format and downloaded onto a device equipped with Internet capabilities. Each Web clipping application looks like an individual application in the handheld's application launcher, and users select and run it as they would a normal handheld application. This launches an internal web browser with the web clipping, which can display static pages or receive dynamic information from an outside web server.

Development of a *Web clipping* involves:

- Design the HTML template, optimizing it for over-the-air transmission to and display on the Palm VII organizer. Design, layout and link HTML files and graphics for the PQA and compile them with QAB.
- Create the server-side scripts to generate HTML clipping pages when appropriate queries are received.

PQA and clippings pages are authored in a subset of HTML 3.2. We can use simple tables, gray-scale color, limited font markup, lists, and images. Referencing HTML pages and graphics is very similar to normal Web pages. One can link to local or Internet files, other PQAs, and other Palm VII applications. Palm also provides several HTML extensions that enable Web servers to send content to both standard browsers and Web clipping-enabled handhelds.

Every PQA should have a top-level or "index" page. This is the first page displayed when the user launches the PQA on the Palm VII organizer. Hyperlinks to other HTML pages and graphics in the PQA and on the Internet start here. There are "Single-Page PQAs" and "Multi-Page PQA". A PQA could easily contain only one on-device page: a query form with its bare minimum of supporting graphics. If the result page to the user query could supply all the needed information, the PQA structure is very simple. The index page is a form that requests one page for downloading and display.

The PQA structure could be more complicated, with multiple pages and graphics on the handheld computer. The PQA can display multiple pages before a clipping is requested and retrieved. The results pages can link to local pages and graphics as well as to Palm OS applications and remote hyperlinks. In this case the users will not want to drill down through

too many layers to find some information. Once users drill down, they will want to make their way back up the information hierarchy, either to a previously visited page or to the top-level index page. The Web Clipping application automatically supplies a back-arrow button. We should supply home buttons and other links to make navigation more intuitive. The History list is a useful way for the user to access cached clippings. And each clipping can include a special META tag to provide a custom string that helps identify the clipping in the History list. In any case, we should strive for ease of navigation and let the shape of the content determine the relationships between the pages.

We will present further some of the *PQA design goals* [7]. As we want to keep static information in the PQA on the Palm VII organizer and dynamically generate clippings on the Internet server, the information flow partitioning is a crucial step in designing the PQA. More stable information should be included in the PQA and installed on the Palm VII organizer. The query form, graphics files, and frequently consulted but nonvolatile information should typically be made part of the PQA. More volatile information should be stored on the Internet server for downloading and display only as requested on the Palm VII organizer. Information flow should always be viewed from the user's point of view and should be based on the constraints and purposes of a handheld computer accessing the Internet over an expensive, low-bandwidth radio connection

The design should strive for clarity, consistency, and utility. A good interface design, even more important when developing content for the small, gray-scale screen of the Palm VII organizer, should have the following *goals*:

- **"Consistent Look and Feel"**

  All pages should have a consistent look and feel, enabling users to easily navigate between pages and across applications, by maintaining their sense of context: similarity in pages create a feeling of predictability, but remaining in the same time aware of the difference between exploratory Web browsing and PQA use. Whenever possible, design elements should be similar to those used in other Palm OS applications. There will be remote links to content on the Internet. The look and feel of the result pages should match that of the query pages, providing a difference between remote and local pages and links.

The pages should be well organized, easy to read, should keep functionality simple, should say and show only what users need, should present only the most essential information to the user and resort to over-the-air hyperlinks only when absolutely necessary. The most frequently used features should be the most easily accessible ones.

- **"User Control, Query Formation, and Results"**
  Users pay for every byte transferred over the air. The application should keep the connect time to minimum, allowing the users to decide when to initiate wireless connections and to understand when an action causes a wireless transaction by providing label commands and dialog boxes that clearly indicate which ones will send and receive data over the air. The Palm OS supplies over-the-air icons that are displayed when a remote hyperlink or button is rendered and which are not automatically displayed in graphical links. The local, on-device hyperlinks should present additional information, when possible.

- **"Give users the tools they need to formulate precise queries"**
  This will ensure that the results downloaded from the Internet are small in size and to the point. The user should be aware when the content of a clipping gets too large (>500 bytes). Pop-up menus should be used to help users form their queries.

- **"Avoid unnecessarily graphics"**
  Graphics should only be used when they effectively communicate a message or make things simpler for the user, especially over the air, because it slows down the response time. They should be stored locally in the PQA whenever possible.

3.1.3. Sample Code

See Annex 1 for some sample code extracted from [7] regarding the use of Palm-specific META tags.

3.1.4. Advantages and Disadvantages

Web Clipping architecture allows *more generous information displays* and *more flexible user interaction* than typical WAP applications. Screen sizes are large enough to allow multiple pieces of data to be presented at once and accommodate the use of check boxes, buttons, menus and graphics that quickly and efficiently communicate input choices and results. Users can simply tap or touch the screen to drill down from summary information to more detail. Input modes include infrared transfer, note pad, touch screens, Graffiti software and fold-out or attachable keyboards. Web clipping along with WAP incorporates industrial strength security designed specifically for mobile computing.

In the same time the technology has some limitations on GUI design imposed by restrictions on both HTML, and the PalmOS API, in addition to the hardware constraints. The following HTML tags: Applet, JavaScript, Vspace, Sub, Link, IsIndex are unsupported. Javascript, frames and nested tables are not supported as well.

3.2. WAP Framework

3.2.1. Motivation and Objectives

The objectives of the WAP according to [8] are:

- To create a global wireless protocol specification that will work across differing wireless network technologies;
- To embrace and extend existing standards and technology wherever appropriate.
- Define a layered, scaleable and extensible architecture;
- Provide support for secure applications and communication;
- To define application architecture model that is suitable for building interactive applications that function well in narrow-band constraint environments.

3.2.2. Features Description

WAP is a set of protocols and conventions that can be used to create manufacturer independent applications for mobile phones. WAP defines two essential elements of wireless communication: an end-to-end application protocol and an application environment (WAE) based on a microbrowser incorporated into the mobile phone. The layered communication protocol is implemented in each WAP-enabled terminal. WAP takes a client server approach [8]. WAP devices can receive data only over compatible WAP networks. WAP applications and data reside on Web servers. Only limited resources are required on the mobile phone.

WAP programming model (see Figure 3-4) is based heavily on the existing WWW programming model. WAE enhances some of the WWW standards in ways that reflect the device and network characteristics. The microbrowser is used to:

- Make a request in WML for a specific URL;
- Pass the request to a WAP Gateway that retrieves the information from an Internet Server either in standard HTML format or preferably directly prepared for wireless terminals using WML. If the content being retrieved is in HTML format, a filter in the WAP Gateway may try to translate it into WML. The WAP Gateway translates requests from WAP protocol into the Internet protocol (HTTP, TCP/IP). The encoders translate WAP content into compact encoded formats to reduce the size of data over the network;
- Send the requested information from the WAP Gateway to the WAP client.

Because of this architecture, the content the URL references can also be something else than WML, Java classes, for example. Terminals of course need UAs that can recognize and handle the content. For Java content this means that there is an implementation of Java VM and class libraries in a device. The WAE also needs to specify the application context issues for Java. Nevertheless, this could be one method for distributing UIs to the WAP enabled devices involving a lot of Java and WAP interoperability issues.

The microbrowser environment coordinates and controls the user interface, being analogous to a standard Web browser. It defines how WML and WMLScript should be interpreted in

the handset and presented to the user. Its specification has been designed for wireless handsets so that the resulting code will be compact and efficient, yet provide a flexible and powerful UI.

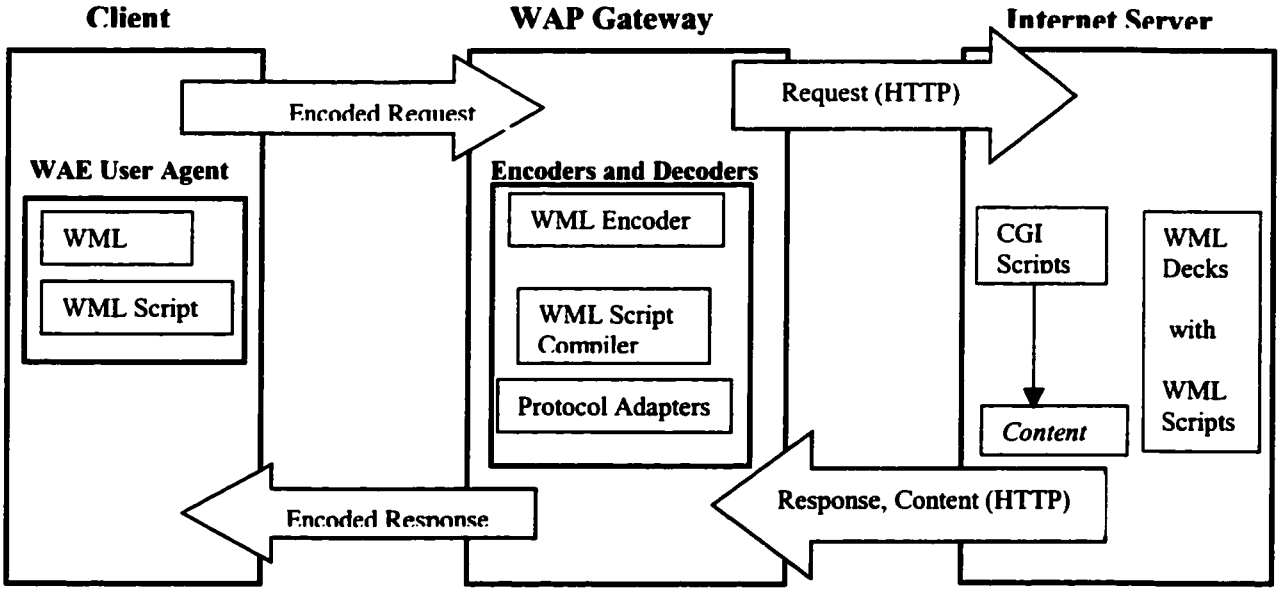

Figure 3-4 WAE Programming Model (Adapted from [8])

The *WAE User Agents* (UA) are client-side in-device software (browsers, message editors) that provides specific functionality (e.g., display content) to the end-user. They interpret network content referenced by a URL. WAE includes user agents for the two primary standard contents: encoded Wireless Markup Language (WML) and compiled Wireless Markup Language Script (WMLScript). User agents have the freedom to decide how to best present all graphical elements within a card depending on the device capabilities.

WAE includes the WML and WMLScript. WML [8] is the language standardized by the WAP Forum intended for specifying UIs for WAP applications. WML is a lightweight markup language, adhering to XML, inheriting technology from HDML and HTML but optimized and designed for powerful applications within the constraints of narrow-band devices. WML describes both content and an interactive UI.

WML uses the "*deck of cards*" metaphor to specify a UI, a powerful and functional UI model that is appropriate for handheld devices. The WML document represents the "*deck*" and is divided into a set of well-defined units of user interactions called "*card*". The cards are logically grouped in a deck and contain the widgets (e.g., a menu or a text field). Instructions imbedded within cards may invoke services on origin servers as needed by the particular interaction. Each card in a deck contains a specification for a particular user interaction. One card is displayed at a time. WML and WMLScript contain methods to change the active card. This enables navigation from one card to another. Specifically, WML contains event bindings, timers and similar anchors than in HTML ('A' tag) that can be used to support navigation.

The *deck*, the unit of content transmission, is similar to an HTML page being identified by an URL. Decks are fetched from origin servers as needed. WML decks can be stored in 'static' files on an origin server, or they can be dynamically generated by a content generator running on an origin server. Users navigate through cards from one or several WML documents with up and down scroll keys instead of a mouse. Soft keys allow the user to perform specific operations appropriate to the application context, or select menu options. A traditional 12-key phone keypad is used to enter alphanumeric characters, including a full set of standard symbols. Navigation functions such as Back, Home, and Bookmark are also provided, in keeping with the standard browser model. Applications can map soft keys for easy user input and use special features to maximize the effect of displaying text on the limited screen.

The user interface concepts of WML are highly abstracted from any particular implementation. WML addresses four aspects of UI specification:

- **Presentation and layout of text and images.** The content that can be presented with WML is very limited. The main content is text with simple formatting elements, but in addition small images and tables can be specified, although terminals may not display them at all. Input elements of WML are also very restricted. Input elements the WML contains are input text field with variable formatting constraints and single and multiple selection lists.

- **Organization of information into coherent units** by grouping the cards inside a deck. WML has an *option selection control* that allows the author to present the user with a list of options that can set data, navigate among cards, or invoke scripts. WML also includes *task invocation controls*. When activated, these controls initiate a navigation or a history management task such as traversing a link to another card (or script) or popping the current card off of the history stack. It may for example, bind them to physical keys on the device, render button controls in a particular region of the screen (or inline within the text), bind them to voice commands, etc.

- **Creation of a navigation model** that supports navigation both within and between decks, jumps to anchored links, and captures specified events (through an UI event handling mechanism) that can trigger navigation or script execution.

- **Articulation of a state model** based on the declaration of string parameters that are replaced by values at run time. This is a particular innovation over HTML-style architectures, dealing with the constrained bandwidth. Each WML input control can introduce variables. The state of the variables can be used to modify the contents of a parameterised card without having to communicate with the server. Furthermore, a variable state can be shared across multiple decks without having to use a server to save intermediate state between deck invocations. WML integrates variables in a different way from that used in HTML and JavaScript.

WML includes a variety of technologies to optimize communication on a narrow-band device. This includes the ability to specify multiple user interactions (cards) in one network transfer (a deck), minimizing the need for origin server requests. WML includes other mechanisms to help improve response time and minimize the amount of data exchanged over-the-air such as supporting variable substitution and providing out-of-band mechanisms for client-side variable passing without having to alter URLs.

WAP incorporates no compression techniques for the textual content, although the WML markup commands are compressed. The deck, the smallest unit of downloadable information

in WML is limited to a maximum of 1400 bytes. This means that applications need to be specifically designed to be very code efficient by using templates and variables, keeping information on the server and using the cache on the phone. WML byte code converting defines a (maybe inefficient) compression technique by string tables. With this technique duplicate strings in the WML bytecode are avoided. This minimizes the data to transfer to the mobile client.

WML has elements that specify a navigational result of an event, such as "go," "prev," and "refresh." The "do" element provides a general mechanism for acting on the current card. Its UI representation depends on the implementation of the specific device. The application developer can only assume that the device will provide a unique widget with which the user can interact. In addition to actions specified within scripts, cards can generate events when they are entered, options generate an event when one is selected, and timers generate an event when they expire. Developers can associate actions with each of these events.

WMLScript [8] is a lightweight scripting language, similar to JavaScript, with procedural logic, loops, conditionals, optimized for small-memory, small-cpu devices. It enhances the standard browsing and presentation facilities of WML with behavioral capabilities, provides a convenient mechanism to access the device and its peripherals, and reduces the need for round-trips to the origin server. WMLScript has the ability to check the validity of user input before it is sent to the content server and to generate messages and dialogs locally, so error messages and confirmations can be viewed faster. WMLScript is compiled into byte code before it is sent to the client in order to cope with the narrowband communication channels and to keep client memory requirements to a minimum. WMLScript is fully integrated with the WML browser and can be used to expose and extend device functionality without changes to the device software.

3.2.3. Sample Code

See Annex 2 for code examples extracted from [8] for card/desk task shadowing, WML deck structure and "fieldset" element. There is another example extracted from [17] of a simple

servlet displaying the current date and time on a wireless device when invoked. The JSP version of the same application is provided as well.

3.2.4. Advantages and Disadvantages

The applications written in WML follow an industry standard. An application written in WML will look good on any device that is WAP-compliant. An application can be customized to take advantage of a particular device's capabilities, by using standard HTTP header mechanisms to learn about the device's capabilities.

WAP works well for building applications that deliver small chunks of timely data with limited lifespans, such as stock quotes, weather forecasts and discount offers. It's also ideal for situations where there's a large database of content, such as an online encyclopedia, from which the user requires only a small piece at a time.

WML being an XML-based language, is easy language to learn and highly discoverable for the first time user. By using the existing Internet model as a starting point, the WAP user interface provides familiar functionality for those accustomed with the Web.

WAP provides a common user interface metaphor that is being used by all industry participants. Just as the desktop metaphor is the de-facto standard for applications on PCs, the WAP card metaphor provides a common interface to which all applications can conform.

WAP devices can receive data only over compatible WAP networks. As a result, WAP is inappropriate for mission-critical applications and personal productivity applications that must be constantly accessible. In addition, because all the processing takes place on the Web, WAP applications can be tedious for processes that involve multiple decisions or transactions. Going back and forth through the gateway to the server when an operation has to be done, wastes bandwidth, and if the network connection is lost midway through the process, the task can't be completed and the work may be lost.

All WML content can be parameterized, allowing the author a great deal of flexibility in creating cards and decks with improved caching behavior and better perceived interactivity.

WAP is an attempt to achieve platform independence through the usage of rather abstract UI elements, although the elements are still quite heavily tied to visual presentation. This has already caused problems when implementing WAP to different platforms. One disadvantage of WML's navigation mechanism is that there is no way to arbitrarily navigate to a new card unless there is an explicit link to it from the current card. WML exhibit only a moderate separation of content versus presentation comparing to UIML. Also it is moderately extensible compared to UIML.

Developers can either use separate URLs for their HTML and WML entry points, or use a single URL to dynamically serve either HTML or WML content according to the requestor's browser type. Although it is possible to translate HTML into WML using an automated system, in practice the best applications use WML to tailor the interface to the specific needs of the wireless user. This allows for the best possible use of the handset features, such as soft keys, and provides the best user experience. The most valuable parts of any Web application are typically the unique content it provides and the back-end database interaction, not the particular HTML that was written to interact with the user. WML's basis in XML also positions it well as a future target markup language or automatic content transformation.

Currently, as we saw in Chapter II, there are many usability issues that are limiting the spread of the WAP phones. While in desktop systems we have mainly two well-known browsers with some compatibility, in WAP-enabled phones a number of microbrowsers tend to accept slightly different versions of WML, assume to interact with slightly different phones (e.g., phones with a different number of softkeys) and interpret the softkeys interactions differently.

WML is pretty impressive with its emphasis on specifying the interactive aspects of an application and by demonstrating how a device-focused language can describe an interface without mapping to specific components in the target platform. These aspects apply to Web

Clipping architecture, as well. Both of these platforms have a strong focus on solving problems in a single application domain and in an efficient manner.

WAP enables application development using existing tools, such as existing XML authoring tools as well as many HTML development environments. Standard Web tools and mechanisms such as Cold Fusion, CGI, Perl, ASP and others can be used to generate dynamic WML applications. *WAP applications* hosted on Web servers can be written not only in WML and WMLScript, but also using existing Web technologies like *Java servlets* and *JSP (Java Server Pages)* [17].

3.3. XML Framework

3.3.1. Motivation and Objectives

The XML framework represents the basis for the WML and UIML frameworks. The XML is the base meta-language for describing other languages like WML, UIML and XIML.

Initially XML has been developed to overcome the limitations of the HTML. XML is intended for describing documents, compared to HTML that is more concerned with displaying documents. XML itself does not provide any instructions on how the document is to be presented or displayed.

The main problem with HTML and WML, besides the fact that they have only a set of predefined tags, is that there is no clear separation between content and presentation. The XML solves this problem by providing a standard for the semantic markup of content. One of the purposes of the XML is to export content in an application-neutral form from legacy systems and transfer it across the wireless network. When the client-side application receives the data, it uses its own presentation language to display the content appropriately for the handheld.

3.3.2. Features Description

XML is a family of technologies including the XSL, a language for expressing stylesheets, and the XSLT, the XSL transformations language for XML documents. XML is the universal format for structured documents and data on the Web. XML is a way of representing and expressing data in a platform independent, extensible, self-descriptive and self-consistent way. It defines a document creation, processing and exchange format.

XML formatted document is a hierarchical collection of elements. The beauty of XML is that we can define our own element types and associate our own semantic with it. DTD is the set of grammar rules, the syntax, for specifying the structure of an XML document. WML and UIML all have their own XML DTDs. A document that respects the formatting rules is "well-formed" and can be parsed by generic XML parsers. XML can be manipulated, though, with scripting languages such as Perl, Tcl, or JavaScript. It can also be used with more-powerful languages such C, C++, or Java. Java has APIs for building XML-based applications: JAXP for XML Processing, JAXB with DOM and SAX for XML Binding & Processing. With JAXB we can compile document schemas -the various fields that go into a document-and generate Java objects. At runtime, the JAXB API automatically converts documents into Java objects and vice versa.

XSL includes an XML vocabulary for specifying formatting. XSL specifies the styling (presentation) of an XML document. XSLT describes how the document is transformed into another XML document that uses the formatting vocabulary. XSLT is used as part of XSL or independently of XSL, for rearranging, adding or deleting tags & attributes. XSLT consists of two parts: a language for transforming XML documents and an XML vocabulary for specifying formatting semantics. Content written in well-formed XML can be automatically translated into content suitable for either HTML or WML by using different XSL style sheet. By applying different stylesheets to the same content we can produce output suitable for display on a variety of devices as shown in Figure 3-5.

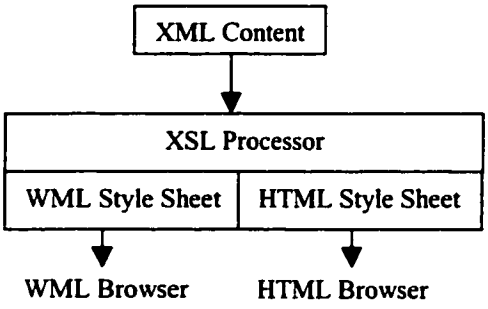

Figure 3-5 Content Transformations

XML offers a *common representation* for *interfaces* executable by multiple devices, taking into consideration the devices constraints, due to the following properties:

- *Platform independent;*
- *Declarative*: through the usage of XSL;
- *Consistent*: through the usage of DTD;
- *Unconventional I/O* : XML can describe unconventional I/O as in WML;
- *Rapid prototyping*: using a stylesheet we can see the results immediately in a browser;
- *Constraint definitions*: XML can contain constraint definitions for the form of the XML itself and for the external resources as well.
- *Easily extensible*: because XML is a metalanguage it is by nature an extensible language;
- *Reusability*: it is relatively easy to fit an existing piece of XML into another.

We propose XSL for describing the user profile. Using the XSLT it is possible to filter out the interested parts for the user. We can make an XSL, which describes the kind of functionality the user is interested in. Letting the XSLT work on a XML structure results in another XML structure only showing the contents of those parts the particular user is interested in. An important part of this conversion is only selecting the relevant paths in the XML user description. This can be done using XPath: an implementation to locate particular branches in the logical tree XML presents. Using an XSL structure we filter out the appropriate section out of the XML description of the user interface. Such an XSL structure can be made by hand, or by monitoring the user's action by an agent, which automatically generates a profile. XSL has declarative properties, which allows us to easily describe what we want, rather than how it has be done.

We will present a *proposed solution* that automatically restructures an existing Web page for specific platforms using XML and XSL [9,10] by transforming it to the device capabilities and user preferences. The system will divide the existing page into smaller pages and re-define the navigation mechanism through the new generated pages. There will be two main steps:

- **Web Page Segmentation**: It divides the document in different sections from contextual, structural, and semantic points of view and depending on the document structure, layout, discontinuities, fonts, styles. The input for the system may be HTML, or XML/XSLT. After the document segmentation we have to generate the ontology of the page. In a typical web page we find different elements: main content, navigation elements, images, etc. The system parses the page and identifies and classifies the different elements in the web page, and the relations between them in order to segment the document. The ontology represents an abstract model of the web page.

- **Small Pages Generation**: Using the device capabilities description (screen resolution, number of color or gray-levels, bandwidth, processing power, languages supported: HTML, WML, XML, input/output capabilities) and the user characteristics description (expertise level, skills, preferences), the system could generates new web pages adapted to the device capabilities and user preferences, by applying a set of XSLT transformations to original page.

  The system builds then a navigation strategy, which represents how we are going to show the information to the user, which information we can remove, and how the user can navigate between the different elements in the page. It also optimizes how the information is transmitted and displayed to the device-user, considering characteristics of the document, devices features and limitations, user knowledge and preferences, etc. Using the navigation strategy the system generates a new page or set of pages with the necessary navigation elements. The following XSLT transformations need be applied sequentially to the original page:

- **HTML Code Adaptation**: Some of the tags or attributes are added, removed or replaced to accommodate the device limitations and constraints.
- **Layout Adaptation**: The layout and distribution of the different elements in the page is modified to fit the device screen. For each of the sections we can put the full content of the section or just the title of that section as an anchor link to that section. Each section can be displayed with different levels of detail.
- **Images and Multimedia Adaptation**: Some graphics are scaled, or the number of colors is reduced according to the device capabilities and user preferences. Other graphics are replaced by their Alt attribute, a link, a simplified version, or simply removed (e.g. decoration graphics).
- **Navigation mechanisms:** Code for navigation is added to each small page to enable the user to navigate through the different pages. Also the system may generate index pages with links to the different generated pages.

One of the benefits that we see for this kind of system is that new transformations can be easily added to the system.

3.3.3. Sample Code

See in Annex 3 an example code that performs an XSL transformation on an XML file with generation of HTML content.

3.3.4. Advantages and Disadvantages

XML technology is an essential element of the wireless picture because with XML the delivery of data to a wide range of devices can be achieved. XML documents are used as a cross-platform way of mapping and communicating data between systems. XML is a promising technology for the platform independent generation of UIs.

The *advantages* of using an extensible markup language (XML)-based language for describing UIs are:

- The separation of UI description from content, by providing a way to specify how UI components should interact, and spell out the rules that define interaction behavior;
- The support for rules to divide responsibilities among different pages, forms, and dialogs;
- The abstraction level that provides for adapting the UI to a particular device or set of user capabilities.

Additionally, XML is much more powerful *than HTML*, specifying individual pieces of information through the use of tags, ability HTML does not provide. Whereas HTML works for presenting information to users, who can scan the page to find what they're looking for, XML can present the like to an application that needs this specific, discrete information. Unlike the case of HTML, element names in XML have no intrinsic presentation semantics.

Modern application servers are now typically configured and managed with XML files. A key point is the usage for defining a format for configuration files. XML provides a good mechanism for maintainability and compatibility between different systems and successive generations of the same system. The use of XML for common, persistent files formats is a big win.

The XSL provides a powerful mechanism for the dynamic transformation of well-formed XML providing the following benefits:

- Without it, a processor could not possibly know how to render the content of an XML document other than as an undifferentiated string of characters. XSL provides a comprehensive model and a vocabulary for writing stylesheets using XML syntax;
- Even XSL was built on CSS, introduces a model for pagination and layout;
- Scrollable document windows and pagination introduces new complexities to the styling of XML content. There is a correspondence between a page with multiple regions, such as a body, header, footer, left and right sidebars, and a Web presentation using "frames". XSL handles the distribution of content into the regions

The combination of XML and context sensitive XSLT allows development of device independent web pages. Pages can be made multilingual by ensuring that XSLT stylesheets

are language independent through the use of vocabulary repositories.

XML is the preferred technology in many information-transfer scenarios because of its ability to encode information in a way that is easy to read, process, and generate. Another advantage of XML is its ability to facilitate an application architecture in which nearly all of the heavy-duty processing is done on the server side. This makes it a natural fit with Java, which has demonstrated its value in server-side application logic. This is a crucial point for enterprise applications, because the servers must be maintained and evolved through multiple generations of client devices. The server-side logic must produce the right data or document for multiple platforms and generations of clients. In addition, both XML and Java are portable, enabling XML data and Java code to be sent to any type of device. Both XML and Java can run on modern clients, such as the recently released J2ME and MIDP.

Because both Java and XML are completely platform-independent, they are used to create Web applications where different clients, different devices, consume and generate information exchanged between different servers that run on varied system platforms. XML and Java offer a very powerful combination data and application independence.

3.4. UIML Framework

3.4.1. Motivation and Objectives

UIML is an XML based language that permits a declarative description of a user interface in a highly device-independent manner - allowing portability across devices and operating systems - and then use a style description to map the interface to various operating systems and devices. One of the reasons for developing the UIML language was to overcome the limitations of the existing languages designed with inherent assumptions about the type of user interfaces and devices for which they would be used. The idea behind UIML is to develop an open-source language that people can use to build interfaces for on web applications, standalone computer applications, cell phones, and elsewhere.

UIML was supposed to be a new declarative language, powerful enough to describe any user

interfaces that before only an imperative programming languages or toolkits (e.g., C with X-windows, C++ with MFC, Java with AWT/Swing) could have implemented. The intention was to bridge the gap between HTML, which allows easy design of UIs with limited interaction, and imperative languages such as Java, which allow design of rich UIs but only in the hands of experienced programmers. In contrast with a declarative language, imperative languages such as Java or JavaScript describe how to implement the functionality of the interface.

According to [11, 12] the UIML language design goals are:

- *"To allow individuals to implement UIs for any device without learning languages and APIs specific to the device"*
- *"To allow non-programmers to implement UIs"*
- *"To permit rapid prototyping of UIs"*
- *"To represent UIs for any device, using any language, any UI metaphor, and any operating system and to permit rapid prototyping of user interfaces"*
- *"To provide one syntax to describe presentation, user interaction with the UI, and connection to the things outside the UI, namely data sources and application logic, regardless of the target device or language"*. Consequently, it is relatively easy in UIML to create a generic description of the User Interface (UI) being developed that can be transformed to various platforms.
- *"To provide a natural separation between UI code and application code logic"*
  The UIML split content from appearance by the means of an device independent UI definition that specifies UI content, and an device dependent style sheet that guides the placement and appearance of user interface elements.
- *"To allow the language to be extensible to support future technologies"*
  UIML tags can be assigned the attribute class. UIML authors can create new values of the class attribute to extend UIML for appliances with interface technologies not in use today. In addition, the style sheet maps values of the class attribute to particular renderings for particular appliances. Events that arise for user interface elements are not hard-wired into UIML. Instead, events are named with a class attribute, and attribute

values are mapped to events appropriate for specific interface technologies through a style sheet.

- *"To reduce the time to develop user interfaces for multiple device families"*
  When authoring multi-platform UIs, we design a family of UIs (see Figure 1-1, Figure 1-2, Figure 1-3). Each UI exposes a different subset of functionality to a user from all the available functions of an application. In describing the UI, time is saved if elements common to multiple devices can be "factored out" and described once in a UI language. Consequently, UIML itself presumes that a family of UIs can be represented by a tree, with the description of functions common to all elements at the root, and differences between different models of devices the leaves. The tree can have an arbitrary number of levels; for example there can be a level representing different categories of devices (e.g., cell phone vs. desktop vs. PDA).
- *"To facilitate internationalization and localization"*;
- *"To allow efficient download of user interfaces over networks to Web browsers on client machines"*. UIML achieves the ideal of allowing the flexibility of downloading code (e.g., Java or Active-X), but using smaller files that are relatively quick to download and that can be cached. For example, with UIML one can create, to our knowledge, any user interface that one can create using Java with the AWT or Swing toolkits, yet require no download of Java code to the Web browser. In essence, the Java portion is downloaded once to the client as a UIML interpreter plug-in, which can then interpret any UIML file.

The most persuasive argument to use UIML is to facilitate multi-platform authoring tools. A vocabulary of generic UI components, properties, and behaviors for multi-platform editing has been defined [22]. We present more about a multi-platform framework using UIML in section 4.2. A multi-platform authoring tool is under development at Harmonia [11], based on a research project at Virginia Tech. The multi-platform authoring tool goal is the mapping of a single UIML file to any target UI language.

3.4.2. Features Description

UIML distinguishes between the UI elements that are present in the UI, the structure of these elements, how are they organized for a family of similar devices, how the interface is to be presented or rendered using CSSs, and how events are to be handled for each UI element. It does not use any metaphor dependent tags to describe an interface.

There are 2 important aspects of UIML: the ability to generate user interfaces for different platforms (see example for Java AWT, WML), and the ability to construct a library of reusable interface components. UIML comes with different rendering programs allowing the generation of user interfaces on different platforms. Also up to now the generation for Java AWT/SWING, HTML, WML, PalmOS, VoiceXML are available.

Runtime interaction is done using events. Events can be local (between interface elements) or global (between interface elements and objects that represent an application's internal program logic (the backend). Since the interface typically communicates with a backend to perform its work, a runtime engine provides for that communication. The runtime engine also facilitates a clean separation between the interface and the backend.

UIML represents an interface in five parts: Interface Structure, Presentation Style, Content (text, images, sounds), Behavior (actions taken in response to user interaction) and Interaction of the UI to the application logic or target UI toolkit objects. An analogy could be made between UI architecture and the Model (Content)-View (Interface Structure, Presentation Style, and Interaction)-Controller (Behavior):

- **Structure:** It defines the overall structure of the interface. The UI description is an enumeration of the set of interface parts, with which the end user interacts and comprising the interface, given in a hierarchical form, to designate the logical structure of the interface. Multiple hierarchies or structures may be given for different families of devices and categories of end users.

- **Style:** Because the availability of specific UI widgets varies from device to device, this element specifies device-specific presentation style for each interface part (similar in principle to CSS), mapping (rendering) the part into the associated widget. It uses vocabulary of names of UI widgets in the platform the UI will be mapped to (e.g., a scrollable selection list). For example if the target is *Java AWT*, the vocabulary might consist of the *java.awt* and *java.awt.event* class names, such as *Frame*, *Menu* and *Button*. If the target is *WML*, the vocabulary might be tag names, such as *card, select, input*. The vocabulary of target platforms is not part of the UIML. It only appears in UIML as the value part of attributes in UIML.

  The style mechanism provides a level of abstraction over the *connection* between parts and the controls on the target device that implement the parts, allowing for the creation of multiple presentations to be defined for a single device. Within a style specification, the rendering property plays a key role in enabling platform independence. This property maps a particular UIML part, event, or call to a widget, event, or method in a particular UI toolkit. Thus a developer can define generic (or functional) event names in UIML and map them to specific control-based events such as menu choices, button clicks, or voice commands in the target UI toolkit.

  In practice, vocabularies will be defined externally for common UI toolkits and incorporated into UIML specifications by reference to a URL. This architecture enables ongoing creation of mappings between parts defined in UIML documents and additional UI toolkits that may become available. The only shortcoming to this model is that it fails to provide an explicit mechanism for creating anything more complex than a simple one-to-one mapping from parts to components in the target toolkit.

- **Content element:** Unlike other markup languages, a UIML document specifies the content (e.g., text, sounds, images) of the interface in a separate XML element. This facilitates internationalization and customization of the UIs for users of varying expertise. The content is then assigned to the interface part by a reference to the content descriptor. The content can also be set programmatically by the application code behind the UI.

- **Behavior element**: defines events for the UI and associates them with interface parts.
  A unique aspect of UIML is that events are described in a generic device-independent fashion and mapped to actual device events at runtime using the style sheet (Unlike style sheets used with HTML). A generic event can be triggered by the user (e.g., when the user enters some text), by the application (e.g., when the backend displays data), or by the underlying system (e.g., when a timer expires or an exception is raised.

  The behavior element contains simple rules, each of which consists of a condition (typically the firing of an event) and a consequent action. An action may result in a change in a property value, or it may invoke a function in a script or in an underlying application library.

  In UIML, the application logic is represented as a collection of components with a well-defined programming interface. These components can be internal (e.g., scripting embedded into a UIML document) or external (e.g., executable code on the local machine or on a remote server). As one of UIML's goals is to break the dependency between the interface and the backend, no calculations are defined in the interface and all calculations and backend integration actions are completely hidden. This is achieved by having a runtime engine monitor all events. Application/interface integrators map generic interface events to scripts or application methods.

- **Peers element:** UIML being a device independent language does not contain any tags that are specific to any particular toolkit. Instead, it used generic names like part, and property. The peer part of the interface maps these generic names to more specific names of the platform for which the UI is developed. It specifies what widgets in the target platform and what functions or objects in scripts, programs or in the application logic are associated with the UI. The peers element provides mappings from abstract interface elements to actual device-specific components that can be rendered (presentation) or executed (logic).

We will present further some UIML *strengths* vis-a-vis of HTML:

- The interface content is not embedded in UIML. The UI definition in UIML contains references (or variables) that will be populated with content when the document is rendered for the particular device. This contrasts with HTML, having no notion of variables, and the cumbersome way of using a scripting language to insert content into an HTML document.
- Unlike HTML, UIML permits a variety of event handling to be described within UIML without relying on a (procedural) scripting language. Each UI component can be associated with a set of events. Each event is able to set a new value for an UI attribute or to invoke a function or method outside the UI. This allows many common events to be handled without any procedural code.
- A UI rendered on a monitor sometimes consists of hundreds of screens with similar appearance but different content (e.g., an interface for a hospital that serves many different specialists in the hospital). UIML has a concept of *UI templates*, which allows one to design reusable interface components, and then customize them to particular uses. This saves a lot of work in building the interface software, and allows UI creation to be partitioned between interface designers and content providers.
- HTML can be embedded in UIML, so those Web pages can appear as part of an application interface.
- UIML is user extensible. Arbitrary interface components (e.g., Active-X controls, Java Beans) can be used with UIML.

We discuss further the two ways in which the UIML interfaces are deployed to the user [12] (see Figure 3-6). The *Interface Server* combines the UI definition, style sheet, and content from a database into a UIML instance. The *UIML Renderer* maps a UIML instance into an arbitrary language and API, such as WML, Windows MFC API or Java.

The Interface Server can deliver the UI to the device in two ways. It could retrieve or generate the UIML document and deliver it to the device. The device in this case is supposed to have the necessary capability to render it for displaying and for the user to interact with.

This is what happens in the case of normal Web browsers on PCs in which the Web page is sent to the browser, which interprets it.

In the second case, the interface server translates the UIML document to the device's native language (WML or BinaryWML for a cellular phone) using the Renderer, and delivers the compiled UIML document to the device which simply, displays it. The server uses the *User Profile* to customize the UIML document, if needed.

The advantage of this second mechanism is that the Interface Server can use a single UIML document and translate it for any device based on the user profile. This also cleanly separates the structure of the interface from its presentation. In either case, a new Renderer and style sheet can be written when a new interface technology is invented to present existing UI definitions.

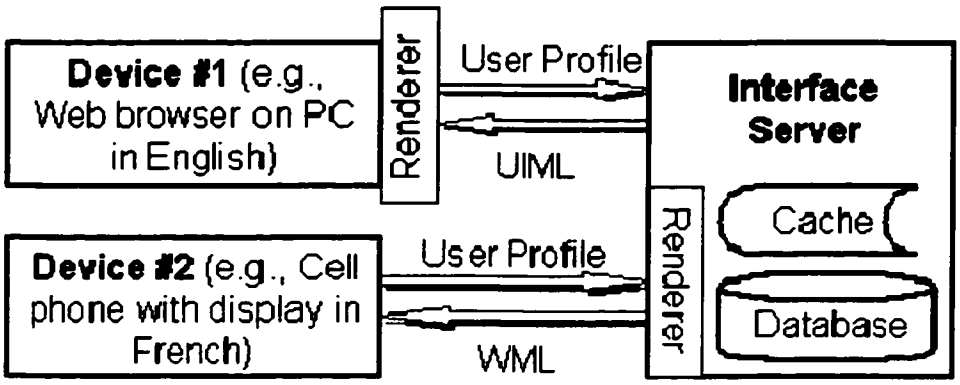

Figure 3-6 Model for UIML Deployment ([12])

Compilation on the server side is mandatory for devices that cannot download applications, such as cellular phones, and where memory is limited. Interpretation is more flexible; for example the Java interpretive Renderer permits the entire UIML interface to appear as a Java bean, so that the interface may be manipulated programmatically by the application logic.

UIML can also be used without a server; in this case UIML or the compiled code or markup language is installed along with the application logic on end-user devices. At runtime, the rendered interface directly communicates with the application logic.

In UIML there are two ways in which the UI can be tailored to the device/end user. First, using multiple structure/style/content/behavior sections gives several alternatives to the user. At runtime, the user may specify the names of each section they want to use (e.g., a simple structure with a graphical style and Greek content) and get a custom interface. Second, UIML can be dynamically generated from a database. The Interface Server can query the client for a list of device features and/or user preferences and dynamically generate the UIML code for the interface from a database query. The user gets a tailored UI that takes advantage of the technologies supported by the device without overloading the network or the device with unsupported code.

3.4.3. Sample Code

See in Annex 4 a UIML sample code of an Online Banking Application rendered for Java Swing, Java AWT, HTML and WML platforms.

3.4.4. Advantages and Disadvantages

The declarative nature of UIML brings some advantages over the traditional procedural programming languages (e.g., C++ and MFC), which specify how an interface is constructed:

- A few lines of UIML take the place of many lines of a procedural language;
- It is faster to use than traditional procedural languages;
- It's easy to enforce a consistent style across many interface screens in an application with UIML's style sheets. There's no comparable facility with procedural languages;
- UIML exposes relationships between pieces of the interface that low level procedural code obscures;
- The UIML Renderer looks like an Active-X control or a Java bean to the rest of an application, so UIML can be combined naturally with procedural languages.

We will present further our conclusions based on our working experience with UIML from the language capabilities point of view. We believe that the declarative language still needs improvement in order to comply to some of the UIML goals mentioned in section 3.4.1. We claim that these goals are not entirely fulfilled for the reasons we further present. When

moving a UI from the PC to a handheld device, the code is indeed simpler when using UIML. But is not about just writing a new style sheet and doing some simple editing of UIML; still heavy modification of procedural code is needed. In order to generate the UIML interface for different platforms such as JavaAWT, Java Swing, HTML or WML, a great deal of knowledge of the languages and methodologies used by these specific target platforms are required for the developers. Besides the final UIML code looks very similar to corresponding programming language code. Therefore we believe it is not quite a language for beginners or non-experience users and it cannot really replace hand-coding of UIs. This leads further to the conclusion that the adapting to different platforms could be pretty expensive, as specific device implementation knowledge is needed and a special renderer is needed for each target device.

This is an important disadvantage of UIML, as each renderer needs its own platform-specific vocabulary of the target UI elements such that the output results vary from renderer to renderer. There is a great deal of differences in the vocabularies associated with each language. Consequently, the UI developer would have to learn each different vocabulary in order to build UIs that would work across multiple platforms. Therefore we do not think that using UIML, reduces dramatically the effort to develop UIs for multiple platforms, in comparison to the effort needed if the UIs had to be built using the native language and toolkit for each platform.

We implemented an Online Banking UI in UIML and tried to render this into Java, HTML and WML. This was achieved by implementing 3 different UIML documents, one for each platform. We still have only one language, but for each platform the code has to be heavily modified with specific language aspects. The language is not ready yet for rendering a single and unique UIML document into any language. We cannot simply create one UIML file for one particular platform and expect it to be rendered on a different platform with a simple change in the vocabulary. Still, with a new Renderer, organizations using UIML should indeed be able immediately to use the new technology on existing applications, but at the cost mentioned above.

Some of the languages advantages that we notice were that changing in UI can be done without recompiling, when using Java rendered and the creation of clean HTML and WML code. On the other hand, there are disadvantages such that it is not possible to include images into UIML documents for Java and that the HTML renderer does not support Javascript.

The UIML architecture enables ongoing creation of mappings between interface parts defined in UIML documents and additional UI toolkits that may become available. The only shortcoming to this model is that it fails to provide an explicit mechanism for creating anything more complex than a simple one-to-one mapping from user-defined interface parts to components in the target toolkit. UIML lacks the ability to specify complex rules because it does not have a direct approach for handling complex mappings, such as many-to-one or one-to-many mappings, from developer-defined components to widgets available in the target platform. This is partially solved by using templates.

However, although UIML allows a multi-platform description of UIs, there is limited commonality in the platform-specific descriptions when platform-specific vocabularies are used. This means that the UI designer would have to create separate user interfaces for each platform using its own vocabulary which is defined to be a set of user interface elements with associated properties and behavior.

In the delivery mechanism there is advantage of running an interpreter on the client side, avoiding the need to send bulky executable code to devices. For example, Java applets today are time consuming to load over the Internet. In contrast, a Java interpretive renderer installed on a PC requires only UIML to be transmitted over the Internet, a benefit for network security. UIML is a small text file compared to the equivalent executable code, just as a form in HTML requires less bytes than a Java applet implementing the same form.

The UIML and mechanisms based on XML technology are technical attempts to address the development problem of plastic UIs, but although useful, they do not provide sufficient insight about how the adaptation process could happen.

3.5. JAVA 2 Micro Edition (J2ME) Framework

3.5.1. Motivation and Objectives

J2ME is a framework that enables anytime, anywhere deployment of services, provides Java technology solutions for building devices across the spectrum - palmtops to desktops and delivers the foundation for intelligent and dynamic networked content [13].

3.5.2. Features Description

A J2ME enabled PDA or cell-phone has the JVM (Java Virtual Machine) or KVM (K Virtual Machine) embedded as hardware or software in the device itself.

The J2ME architecture [13] (see Figure 3-7) is modular and scalable in order to support the flexible customizable deployment demanded by the consumer and embedded markets, by providing a range of virtual machines, each optimized for the different processor types and memory footprints. This is how the diversity in form, function and features of the network-connected devices is addressed.

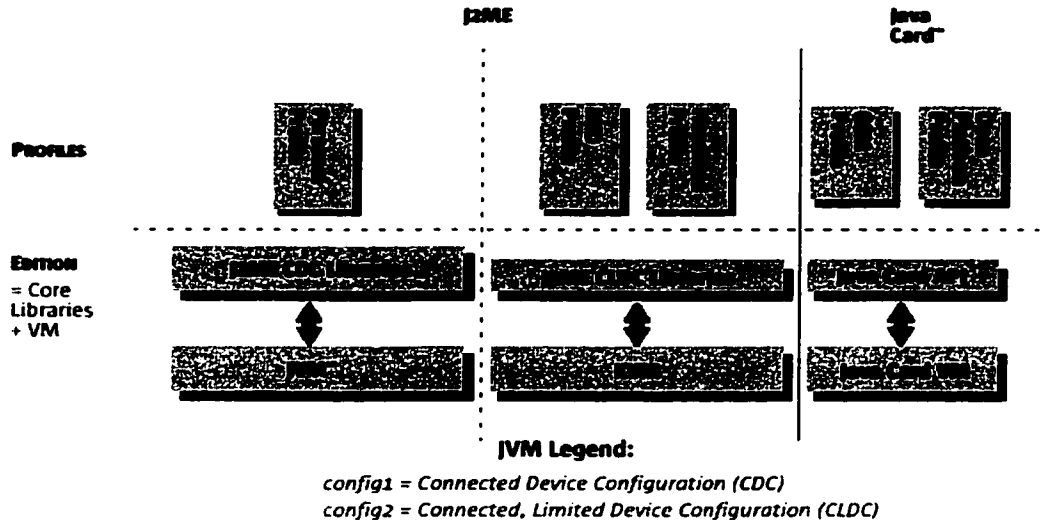

Figure 3-7 J2ME Environment [13]

J2ME supports minimal configurations of the JVM and Java APIs just for the essential capabilities of each kind of device. As device manufacturers develop new features in their devices, or service providers develop new and exciting applications, these minimal configurations can be expanded with additional APIs or with a richer complement of JVM features. The "configuration" and "profile" concepts are associated to J2ME:

- **Configuration.** J2ME is deployed in several configurations. Each configuration defines the minimum set of the Java features and libraries available for a particular "category" of devices, each with similar requirements on total memory budget and processing power, and representing a particular "horizontal" market segment. CLDC (Connected Limited Device Configuration) is one of the J2ME configuration that defines targeted Java platforms for mobile connected information devices with constrained 16-or 32-bit CPU and memory resources between 160 kB and 512 kB. It is composed of the KVM and core class libraries. The CLDC itself does not define any GUI functionality. Application developers and content providers must design their code to stay within the bounds of the Java virtual machine features and APIs specified by that configuration. In this way, interoperability can be guaranteed among the various applications and devices in a particular class.

- **Profile**. A J2ME device profile extends and is implemented "upon" a particular configuration. It defines the minimum set of APIs, available on a particular "family" of devices, being more domain-specific than the class libraries provided in a configuration. Some of these profiles will be very device-specific, while others will be more application-specific. Applications are written "for" a particular profile and are thus portable to any device that "supports" that profile. A device can support multiple profiles. All profiles have the ability to safely download code onto a device and configure the Java environment. Profiles implemented on top of the CLDC address these features: application life-cycle management (installation, launching, deletion), UI, event handling, high-level application model (the interaction between the user and the application). The only existing profile is the MIDP (Mobile Information Device Profile) designed for cell

phones. Applications that run on devices supporting MIDP are called MIDlets. MIDP have libraries for the *user interface*, database, and device-specific networking.

Official GUI classes for the J2ME are included in MIDP profiles. The GUI classes included in the MIDP are not based on the AWT for the following reasons:

- The AWT is designed for desktop computers and optimized for these devices.
- The AWT assumes certain user interaction models. The component set of the AWT is designed to work with a pointing device such as a mouse; however many handheld devices, such as cell phones, have only a keypad for user input.
- While the AWT has a rich feature set, it is mainly desktop-based. Also, the feature set includes support for features not found on handheld devices. For example, the AWT has extensive support for window management, such as resizing overlapping windows. However, the limited display size of handheld devices makes resizing a window impractical. Therefore, the window and layout managers within the AWT are not required for handheld devices.
- When a user interacts with an AWT-based application, event objects are created dynamically. These objects exist only until each associated event is processed by the system, at which time the object becomes eligible for garbage collection. The limited CPU and memory of handheld devices cannot handle this behavior.

GUI classes are included in MIDP. The MIDP UI APIs are logically composed of high-level (implemented by classes that inherit from the Screen class) and low-level (implemented by Canvas and Graphics classes) APIs [6, 14, 15]. The high-level APIs use a high level of abstraction to achieve the software portability across various cell phones and pagers. The trade-off is that the high-level APIs limit the amount of control the developer has over the human interface's look and feel. Interaction with components is encapsulated by the underlying implementation and the application is not aware of such interactions. If an application has to be aware of component layout, scrolling, and focus traversal it compromises portability. The underlying implementation is accomplished by the handset manufacturer and does the necessary adaptation to the human interface on the device's hardware and native user interface style. This ensures a consistent user experience between

the native applications and the MID Profile applications. Typical applications using a high-level API are stock transactions, travel reservations, online purchase of concert tickets, news headlines, weather updates, and traffic information.

On the other hand, the low-level API:

- Provides very little abstraction and requires a bit more design work to remain portable;
- Designed for applications that need precise placement and control of graphic elements, as well as access to low-level input events;
- Allows the application to access special, device-specific features, therefore the MIDlets that access the low-level API are not guaranteed to be portable.

Games are a typical application that use this API.

The actual drawing to the MIDs display is performed by the device implementation. The application cannot access concrete input devices such as individual keys. Because of the small displays, instead of a window system, the central abstraction of the MID Profile UI API is the "*screen*". A screen is an object that encapsulates device-specific graphics rendering user input. Only one screen is visible at a time, and the user can only interact with the items on that screen. The screen handles all events that occur as the user navigates on the screen. Only high-level events are passed on to the application. Screens scroll vertically only; there is no horizontal scrolling. Screens should be simple in design, basically one user task per screen, and contain as few UI components as possible to complete the task. An application is composed of a set of screens, which the user steps through as they complete tasks. The set of screens that compose an application do not need to be linear, branching and jumping between screens to enable proper usability is expected. The application developer should design the application in such a way that navigation is clear and obvious to users.

There are the following types of screens:

- Screens that encapsulate a complex user interface component like "List" (an implicit menu list; a single-choice list; and a multiple-choice list) or "TextBox" (will not support fancy text formatting or font styles and allow only the basic text editing capabilities) and have predefined structure. The application cannot add other components to them.

- Generic "Form" screens. The applications can an arbitrary mix of items including images, read-only text, editable text, date and time fields, gauges and choice groups.
- "Alert" screen.

It is recommended to limit the screen to a single user task. Form screens should be not longer than three to four displays in length. Forms that require extensive scrolling can overtax the performance of the device and can become unwieldy for the user to interact with. In general, any item or its subclass may be contained within a form. The implementation, not the application, handles layout, traversal, and scrolling. None of the components contained on the form screen are able to scroll independently — the entire contents scroll together vertically. When a form is present on the display, the user can interact with it and its items indefinitely, for example, traversing from item to item or scrolling. It is the form screen capabilities, combined with the MID Profile abstract commands, that provide the most flexibility in developing compelling, competitive, portable mobile interactive solutions for the cell-phones.

To ensure application portability across devices, a method is needed to insulate the developer from the actual set of buttons that may be on a device. MIDs will either have a one-handed ITU-T phone keypad or a two-handed QWERTY-style keyboard. Beyond that, devices may have any number of programmable (soft) buttons or no programmable buttons at all. There may be dedicated hardware buttons for Back, Help, Clear, etc., or there may no dedicated buttons at all. The Abstract command API is the layer that is used to map between the buttons on a device and the set of commands that an application developer attaches to a screen.

There is one policy per phone for how buttons are mapped on it. What is possible is to define the set of actions the user can select from each screen in the particular application. Each action is defined as a command with an associated command type, priority, and label. The type, priority, and label provide hints to the underlying device implementation that can be used to map the commands appropriately. For example, a command of type HELP might get mapped to a dedicated Help button on a phone.

Working at this high-level of abstraction enables developers to state their preferences, but offers have no guarantee that the commands will appear in a specific order or on a specific button. Commands are mapped by type first, and then priority within that type. Developers should plan on testing their application on as many different target devices as possible to understand how the abstract command mapping policies of different devices will effect their application.

If an application asks for more abstract commands than there are available buttons, the MID Profile reference implementation, for example, uses another UI mechanism, the Menu, to make the commands accessible to the user. The overflow of abstract commands is placed in a menu and the label Menu is mapped on to one of the programmable buttons.
J2ME provides a set of simple UI components. Programming with these components is easy in comparison with the J2SE AWT and Swing libraries because the MIDP GUI API is simpler.

Motorola's LWT (Lightweight Windowing Toolkit) [19], the first available graphical user interface extension for J2ME technology-enable wireless products, enhances the GUI capabilities of the MIDP platform and gives developers better control over the look and feel of their applications, making simpler, more intuitive system software possible for mobile communication products.

Motorola's LWT is an advanced graphics API for the J2ME. LWT integrates with the LCDUI API within the MIDP, and enhances the capabilities to include a component-level API. Developers can control the contents and layout of their screens, and include components such as buttons, checkboxes, text fields, and images. LWT applications can even mix MIDP screens and LWT screens in the same application. LWT is extensible so developers can add new components, or match the look and feel of components on another user interface. Components such as Button and Checkbox may be tailored to match existing applications on other platforms or match other MIDP compliant applications like MIDlets. In addition, developers can create imaginative new components. LWT integrates with and extends the MIDP without impacting the portability of other MIDlets. MIDP APIs are supported, so

MIDlets are not disrupted. Manufacturers can easily add LWT to a MIDP implementation. The basic requirement is MIDP on the device, and a small amount of flash memory. LWT makes it easy to add graphics, making applications fast and easy to read.

The MIDP for Palm OS [18] is a J2ME application runtime environment based on the CLDC and MIDP specifications. It is targeted at Palm OS handhelds running Palm OS. The product feature sets include:

- MIDP specification features: Low-level graphics API (Canvas), High-level graphics API (LCDUI) Abstract Commands and Canvas Input, Database access (RMS), Networking (HTTP)
- Palm OS specific features: Preferences, HotSync support, MIDlet beaming support.

Developers can use the same tools to develop MIDlets running on Palm OS handhelds as well as on other MIDP compliant devices. In MIDP for Palm OS, the abstract commands will map onto buttons that are shown at the bottom of the screen. All Abstract Commands are also duplicated in the menus for usability reasons. The goal of MIDP is to provide a common set of APIs across mobile phones; therefore no Palm specific APIs are available. MIDP offers a way to develop Java on a Palm OS device.

3.5.3. Sample Code

See Annex 5 for a sample MIDlet code extracted from [16] for displaying a string in display area of the mobile phone.

3.5.4. Advantages and Disadvantages

J2ME allows applications to be dynamically downloaded to a mobile device in a secure fashion. The application can be operated in disconnected mode (e.g. stand-alone game, data entry application) and store data locally, providing a level of convenience that is not available on current browser-based solutions. Because the application resides locally, the user doesn't experience any latency issues, and the application can offer quite a rich user interface (drop-down menus, check boxes, animated icons). The user is empowered to

control when the application initiates a data exchange over the wireless network. This allows for big cost savings on circuit-switched networks where wireless users are billed per minute, and allows a more efficient exchange of data. Additionally, J2ME applications can leverage any wireless network infrastructure, taking advantage of a WAP network stack on current circuit-switched networks.

The MID Profile enriched the cell phone functionality. The interaction of lists, alerts, text boxes, and form screens with high-level abstract commands provides a workable way for developers to write portable applications in the Java that can run across a variety of MIDs. This opens the door to a variety of new wireless services and applications. For example, the device can automatically track a user's stocks and let them know when a specified stock reaches a certain price. A similar application could track auctions, alerting the user when bidding reaches a particular point. Or, a user may wish to play a game that can be downloaded and played locally; scores can then be uploaded to compare against other players. A key benefit to incorporating J2ME technology into a wireless device is that it enables consumers to continually upgrade the applications on that device once it has been purchased.

J2ME complements the WAP with respect to security issues. The WAP server with its decryption/ encryption routine is an easy target for hackers to steal confidential information. Java can provide a more direct route to the wireless device. No gateway or WAP server is necessarily needed, and no decryption/encryption occurs. WAP will continue to be useful for simple, content needs, while Java will be necessary for secure, faster, and more robust features and services. J2ME allows industry groups to define specific Java based profiles or specifications defining a Java based platform suited to a specific class of device.

The key advantages that Java technology provides for mobile development are the portability, simplified support and maintenance, reliability and secure networking. Since the J2ME platform is an industry standard, the same software and applications can work on a variety of different devices.

4. Chapter IV - High-Level Frameworks for Mobile UI Development

4.1. Automatic UI Design Process

Model-based systems for user interface development [2, 3] exploit the idea of using a declarative interface model to drive the interface development process. An interface model represents all the relevant aspects of a user interface in some type of interface modeling language. The model-based systems attempt to produce automatically a concrete interface design (i.e., a presentation and dialog) from the abstract representation of UI (task and domain model). This is done through the mapping process of the abstract elements in the concrete elements in the interface model.

For example, given user-task *t* in domain *d* find an appropriate presentation *p* and dialog *D* that allows user *u* to accomplish t. Therefore, the goal of a model-based system in such a case is to link t, d, and *u* with an appropriate *p* and D.

In the current model-based systems [2] the process of generating the concrete interface involves the stages as shown in the Figure 4-1.

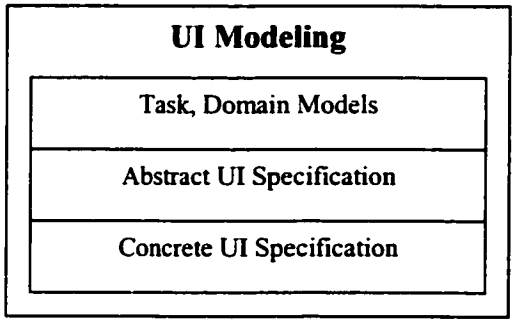

Figure 4-1 UI Modeling

The *task model* represents the tasks that users need to perform with the application. The *domain model* describes the structure of the information that the application provides or the objects that the user can view, access and manipulate through the interface. Tasks models typically represent tasks hierarchically decomposed into subtasks, until the leaf tasks represent operations supplied by the application and information included for sequencing

(and, or, xor). It may include references to the domain objects needed and produced in each task. The task model it is used during the automatic generation to determine the interface dialog and the information to be shown in each window.

The *abstract UI specification* represents the structure and content of the interface in terms of two abstractions, *abstract interaction objects* (AIO), *information elements* and *presentation units*. AIOs are low-level interface tasks such as selecting one element from a set, or showing a presentation unit. Information elements represent data to be shown, either a constant value such as a label, or a set of objects and attributes drawn from the domain model. Presentation units are an abstraction of windows. They specify a collection of AIOs and information elements that should be presented to users as a unit. In summary, the abstract UI specification specifies in an abstract way the information that will be shown in each window, and the dialogue to interact with the information.

The *concrete UI specification* specifies the style for rendering the presentation units, and the AIOs and information elements they contain. The concrete specification represents the interface in terms of toolkit primitives such as windows, buttons, menus, checkboxes, radio-buttons, and graphical primitives such as lines, images, text, etc. In addition, the concrete specification specifies the layout of all the elements of a window.

The automatically design of the interface is done in the following sequence of steps:

- *Determine the presentation units.* Determines the windows that will be used, and what information will be shown in each window.
- *Determine the navigation between presentation units.* Computes a graph of presentation units that defines which units can be invoked from which other units.
- *Determine the abstract interaction objects for each presentation unit.* Specifies the behavior of each element of a presentation unit in an abstract way (e.g., select one from set).
- *Map abstract interaction objects into concrete interaction objects.* The concrete interaction objects represent the widgets available in the target toolkit.

- *Determine the window layout.* Determines the size and position of each concrete interaction object.

The first three steps build the abstract user interface specification, and the last two build the concrete specification. We believe that tools are needed to help designers move from models to concrete user interfaces by choosing from several available criteria. We claim that the limitations of current model-based systems for generating mobile UIs for different platforms are due to the lack of a general solution to the mapping problem for all interfaces and to the lack of availability of a general framework to search for solutions to the mapping problem for individual interfaces. This will be a major concern for our "universal" "high-level" framework.

4.2. Framework for Building Multi-Platform User Interfaces through a Multi-Step Transformation Process

We saw in Chapter 1 how difficult it is difficult to develop an application for multi-platform deployment without duplicating development effort. Here we present a multi-step process of building multi-platform UIs that reduces the duplication of effort by factoring out common parts of interfaces in different levels [20]. The different levels all use UIML as the representation language and provide transformations that convert a UI from one level to the next (see Figure 4-2).

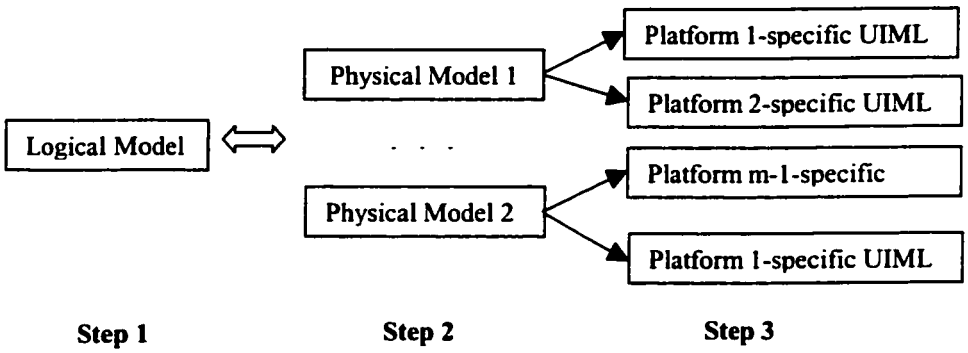

Figure 4-2 Building Multi-platform UIs using UIML (Adapted from [20])

- **Logical model** of the interface is constructed to capture the UI description at a higher level of abstraction than is possible by any physical model, without any knowledge of the widgets or layout used in the physical model. It helps in providing the same functionality of the application on different platforms. The logical model can be transformed into different physical models that are specific to groups of devices or platforms that have similar layout facilities. A hybrid task/domain model could be used that can be transformed to the physical model.

- **Physical model** of the interface is created that captures the UIML description of the interface for a group of devices or platforms that have similar layout facilities. A set of generic vocabulary of UI elements [22] is used, which are not dependent on any particular platform or language and are sufficient to provide general functionality needed for UIs. We can see examples from the UIML Generic vocabulary in Tables 4-1 and Table 4-2. This model describes the hierarchical arrangement of the interface being generated using generic UIML interface elements. The interface can be created at Step1 or Step2.

  Different HTML browsers and the Java Swing platform can all be part of one physical model based on their layout facilities. Some platforms might require a physical model of their own. The physical models that can be built currently are for the desktop platform (Java Swing and HTML) and phone (WML). These physical models are based on the available renderers. The specification for the physical model is already built. The first type of transformation is the mapping from the logical model to the physical model.

  This type of transformation has to be developer-guided and cannot be fully automated. By allowing the UI developer to intervene in the transformation and mapping process, it is possible to ensure usability. Once the user had identified the mappings, the system will generate the physical models based on the target platforms and the user mappings.

- **Platform model.** A transformation algorithm is used to generate platform-specific UIML, from generic UIML. This platform-specific UIML can then be used in

conjunction with a renderer to render the interface. This process can be largely automated. However, there are certain aspects to the transformation that need to be guided by the user. The developer has to select the mapping from elements in the generic vocabulary to one or more sets of elements in the target platform and has the liberty of enhancing the platform-specific UIML to add more platform-specific elements. The way this is currently implemented, the developer does this as a special property of the UI element.

Generic User Interface Element	JAVA™ (Swing)	Palm OS® (Component)	HTML 3.2[1,2] (tag)	WML (tag)
GButton	JButton	Command Button	<input type="button">	<do>
GLabel	JLabel	Label		<p>
GIcon	Icon	Bitmap	<img>	<img>
GSLTextRegion	JTextField	Field	<input type="text">	<input>
GList	JList	Popup List	,	<select>
GDialog	JDialog	Alert Dialog, Process Dialog	-	-
GMenu	JMenu	Menu	-	
GArea	JFrame, Window, JPanel, JScrollPane, JTabbedPane, JTable, JInternalFrame	Window	<form>, <table>	<card>
GTopContainer	JFrame	Form	<html> + <body>	<wml>

Table 4-1 UIML Generic Vocabulary 1 [21]

Generic User Interface Element	JAVA™ (Swing)	Palm OS® (Component)	HTML 3.2 (tag)	WML (tag)
GMLTextRegion	JTextArea	Field	<textarea>	<input>
GCheckbox	JCheckBox	Checkbox	<input type="checkbox">	-
GRadioButton	JRadioButton	Pushbutton	<input type="radio">	-
GComboBoxList	JComboBox	Popup List	-	-
GScrollingList	JList	Popup List	-	-
GTree	JTree	-	-	-
GTable	JTable	Table	<table>	<table>
GMenuItem	JMenuItem	-	<select>	<choice>
GProgressIndicator	JProgressBar	Progress dialog	-	-
GSlider	JSlider	Slider	-	-
GToolBar	JToolBar	-	-	-
GToolTip	JToolTip	-	-	-

Table 4-2 UIML Generic Vocabulary 2 [21]

The researchers conclude that UIML provides a practical approach for the creation of multi-platform UIs through a process of transformation that encompasses different devices with various capabilities. The physical model and the transformation algorithm that converts the physical generic model to the platform-specific UIML has already been developed at Harmonia Inc.[22] and will be incorporated within a multi-platform Authoring Tool for UIML. The logical model will be incorporated within this tool in the future. It's about an IDE (a Single Authoring Tool for Multiple Platforms), which allows graphical compose of UI. The events for each UI Component can be defined without any knowledge of Java, HTML or WML. UIML is generated automatically from graphic design, and can also be edited.

4.3. Framework for Task and Business knowledge Integration

This model-based framework integrates task and business-object knowledge into the development process and its underlying representations [23]. It exploits the user, task and business-object models as the basis for the development of the application logic and the UI design. It is essential to model the users when they have different preferences, abilities, and privileges. It is also often appropriate to model the domain characteristics of the tasks supported by the UI. Such information often guides the selection of appropriate widgets.

The user interface is considered as a simplified version of the MVC model and is separated into a model component and a presentation component. The *model component* describes the feature of the user interface on an abstract level. It is also called *abstract interaction model* (AIM). The user-interface objects with their representation are specified in the *presentation component*. It is also called *specific interaction model* (SIM) representing the features for the specific UI objects for each device. During the development a mapping transforming the abstract interaction model to the specific interaction model is necessary.

The XML allows the description of the abstract interaction model, the description of specific characteristics of different devices and the specification of the transformation (mapping) process from the AIM to the SIM (see Figure 4-3).

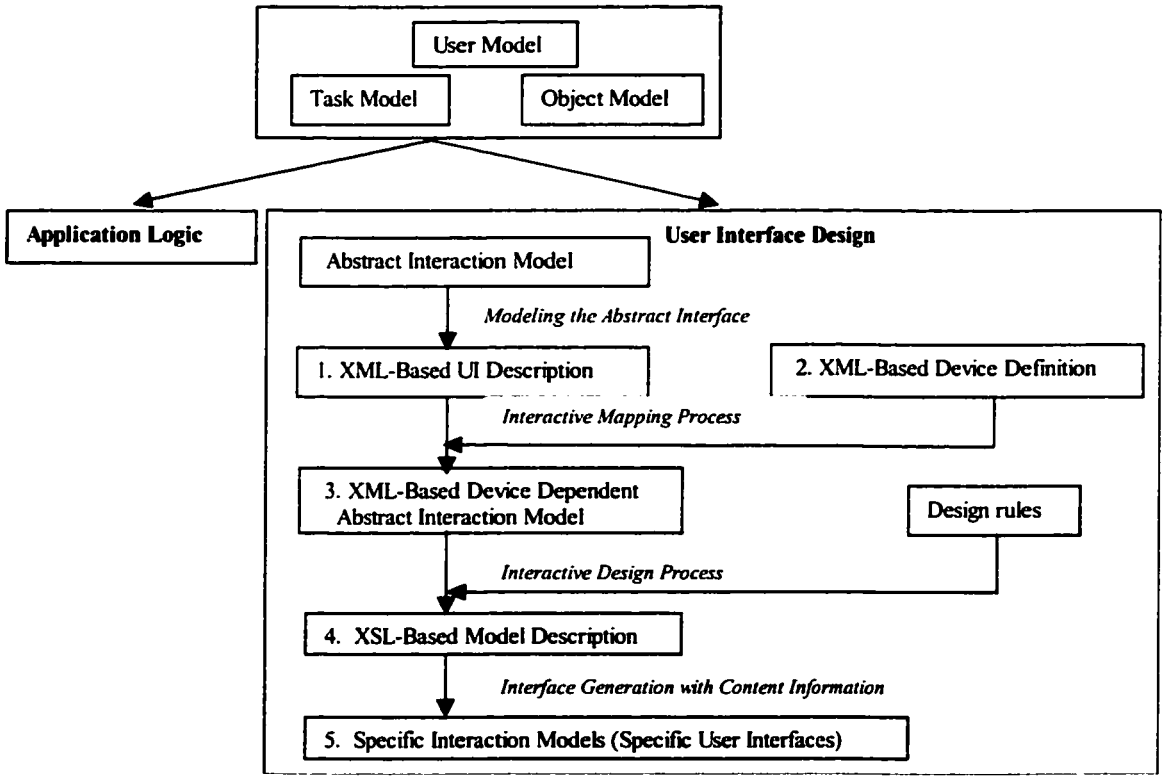

Figure 4-3 Model of the transformation process from an abstract to specific UI (Adapted from [23])

The main steps involved in the transformation process are:

1. **XML-Based Abstract Interaction Model.** The UI is composed at this stage by one or more abstract UIO's each composed by other UIO's or one or more input /output values given in an abstract form and describing features of the UI in an abstract manner. There will be a mapping process of these abstract UIO' to concrete ones. The abstract interaction model is transformed into a notation, which is based on a language of the XML family.

2. **XML-Based Device Definition.** There are XML documents containing abstract descriptions of the properties and specific features of various target devices. They are needed to support the transformation process from an abstract interaction model to a device dependent abstract interaction model. The specifications are not only necessary for this transformation they influence the specification of formatting rules as well.

3. **XML-Based Device Dependent Abstract Interaction Model.** This model fulfills already some constraints coming from the device specification. It uses available features and omits not available services. The result of the mapping process is a file in which all abstract UIO's are mapped to concrete UIO's of a specific representation. The structure of this file is based on a XML-DTD. The file is specific according to a target device and includes all typical design properties of concrete UIO's like colour, position, size. It is a collection of option-value pairs where the content of the values are specified later on in the design process and describes a "skeleton" representation of the specific UI. The user interface is designed on the basis of this skeleton during the following design process

4. **XSL-Based Model Description.** The creation of the XSL-based model description is based on the knowledge of available UIO's for specific representations. It is necessary to know which values of properties of a UIO are available in the given context. The "skeleton" and available values of properties are used to create a XSL-based model description specifying a complete user interface.

5. **Specific User Interfaces.** A file describing a specific UI will be generated by XSL transformation:
   - Creation of a specific file representing the UI (WML, HTML, Java)
   - Integration of the contents (database, application) into the interface.

The generated model is already a specification runable on the target platform. It is constructed by a transformation process, which uses as input the device dependent AIM coded in XML. The UI could be generated once, like for java.AWT or it has to be generated dynamically several times, like for WML. In the first case there is no need for a new generation of the user interface if the contents changes. The content handling is handled within the generated description file. In the second case the content will be handled by instructions within the XSL-based model description. Each modification of the contents results in a new generation of the generated description file. This approach allows a reuse of designed models apart from final representations.

Attempts in applying this approach were done in the TADEUS [27] project, where both AIM and SIM models have grammar representations. It proved that the development of different grammars for different platforms is time consuming. Therefore the transformation process needs to be improved especially at the integration of the specific features of different devices.

This approach shows that XML is a promising technology for the platform independent generation of UIs. XML contributes to the model-based UI design approaches in a way that it is used for representation and transformation features ensuring a flexible development process from an abstract interaction model to a specific interaction model.

4.4. Framework for Context-Sensitive UIs emphasizing the Task Modeling

This framework identifies four major steps for designing context-sensitive UIs [24]:

- **Production of a context-sensitive task model.** This task model should foresee all sub-tasks that might be considered in a single context of use or in multiple contexts of use. This activity may involve adding sub-tasks that are especially supported in a particular context of use and withdrawing of sub-tasks that are not supported in a particular context of use. This task model should obey to the separation of concern principle: the non-context-sensitive part should be clearly separated from the context-sensitive part, while showing relationship between them.

- **Production of a generic UI model**. This generic UI model is supposed to model a UI independently of any context of use, thus considering the non-context-sensitive part of the context-sensitive task model.

- **Production of a specific UI model**. This specific UI model is supposed to model a UI depending on the constraints of intended contexts of use, thus considering the context-sensitive part of the context-sensitive task model. Due to variation of these constraints, multiple UI models may be produced. Sometimes one model tries to cover as many contexts of use as possible, provided that the usability properties are preserved and that a same UI can hold the expected properties simultaneously.

- **Production of a final running UI.** The previous model can then be exploited at design time to automatically generate code (e.g., in HTML with CSS cascading style sheets for Web-based contexts, WML for cellular phone contexts, in XML with XSL for XML-compliant browsers), for a particular context of use or be interpreted at run-time to produce the expected UI.

Figure 4-4 highlights that some approaches or languages are more targeted at some steps of this method. For example, UIML does not encompass any task model, but precisely supports a smooth transition from a platform-independent UI model to a platform-dependent UI model, which is in turn converted into code of a programming language of the target computing platform. XIML may serve as a continuous model repository for all steps, including the last one if a run-time interpretation is performed. Any XML-compliant may serve equally to model aspects in the steps. XIML aims to become an industry-wide framework in which case it will ensure the performance, security, and maintenance of this approach to interface design.

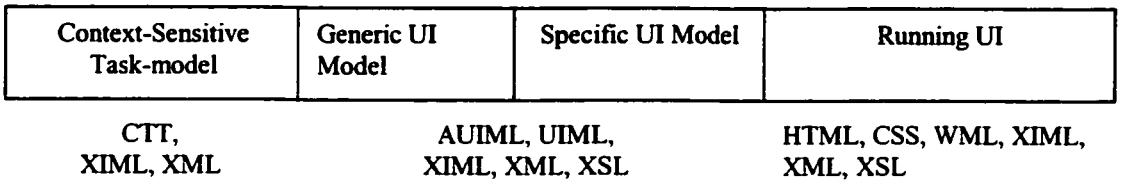

Figure 4-4 Designing Context-Sensitive UIs (Adapted from [24])

This framework proposes a few possible approaches for task modeling for context-sensitive UIs, all of which exploit the concept of *unit task* and *factoring out of the commonalties across multiple contexts of use.* For a unit task a clear separation of concern between the context sensitive part and non context sensitive part is done. In the task tree model a new decision tree which branches to context-sensitive tasks (a task that may require at least one context of use switching for its performance), depending on contextual conditions.

The *monolithic, graph oriented, context-sensitive separation, complete separation* approaches are considered for extending the existing tasks modeling techniques using the CTT (ConcurTaskTree) notation to support the context-sensitive tasks [25].

The *monolithic approach* consists of drawing a global task model that directly encompass non-context sensitive parts and context-sensitive parts. In these parts, we can find every level of the task decomposition into sub-tasks and so forth. As the context of use may occur wherever in the task accomplishment, the separation between non-context-sensitive and context-sensitive task units may be located everywhere in the task tree. The main advantage of this approach is the unicity of the task model as everything is modelled into a single model. However, within context-sensitive parts, it is hard to differentiate nodes branching to sub-trees to reflect possible changes of context from nodes indicating task units to carry out after the context of use changed.

The *graph oriented approach* tries to address this shortcoming by extracting sub-trees resulting from the context-sensitive part into separate trees that can be related where needed. It consists of establishing a relationship to all sub-trees describing task units resulting from changes of context. The biggest advantage of this approach relies in its distinction between non-context-sensitive and context-sensitive parts. However, the resulting model looses its hierarchical structure to become a directed graph. This approach is introducing additional relationships that may increase model cluttering. This kind of graph is also harder to interpret and to manipulate by a tool. Conversely, no representation of how and when the context of use may change in the task model is given.

The *context-sensitive separation approach* (see Figure 4-5) attempts to solve this problem by recognizing that the context-sensitive part actually holds two types of arcs and nodes:

- Traditional arcs and nodes as one can find in a classic task tree: these nodes represent the task units at their various levels of decomposition and the arcs express their parent-child relationship.

- Decision arcs and nodes so as to select the particular sub-tasks to carry out, depending on conditions expressed on properties of the context of use. These properties are called contextual properties, such as, for example, the type of computing platform, the screen resolution, and the network bandwidth. Conditions on these contextual properties are

called in turn contextual conditions. The chaining of contextual conditions form a decision tree that properly branches to the appropriate task units, depending on the context of use selected.

This approach makes clearer the distinction between the non-context sensitive part, the context sensitive part, and the decision tree making the link between those parts:

- A *non-context-sensitive tree* containing the decomposition into tasks units that do not change if the context of use changes which ends up with decision points, where to branch to a decision tree for considering right context;
- A series of *context-sensitive trees* modeling task units for all supported contexts of use;
- A *decision tree* summarizing all contextual conditions

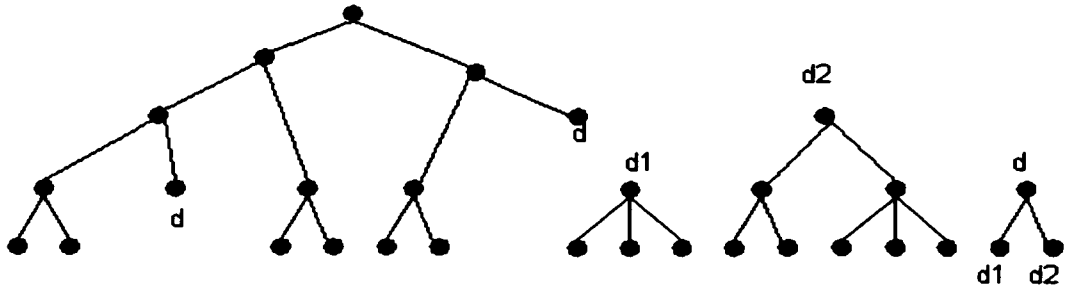

Figure 4-5 Context-sensitive separation approach [24]

However, it is likely that different contexts of use may be considered similarly at different locations of the non-context sensitive part. There is consequently no way to factor out the common parts of the context-sensitive part, depending on the selected context of use.

The *complete separation approach* (see Figure 4-6) aims at factoring out similar sub-trees, which are possibly used at different locations in the global task tree, thus minimizing duplication of these context-sensitive sub-trees. It consists of identifying:

- A *non-context sensitive part*, which similarly produced by the context-sensitive separation approach;
- A series of *separate decision trees* representing branching to all the possible contexts of use with decision points as root;

- A series of *sub-trees of factored task units* to carry out when needed, one sub-tree for each considered value of the final contextual condition. The difference with respect to the previous approach is that this context-sensitive part consists of a collection of sub-trees that may be called from different places in the decision tree. Therefore, each such sub-tree is unique.

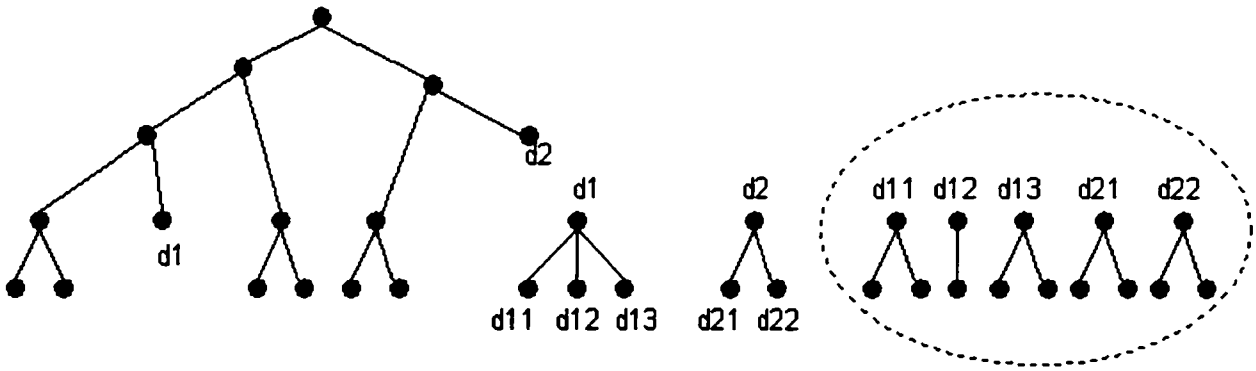

Figure 4-6 Complete separation approach [24]

The advantages of this approach is the more obviously separation than in the context-sensitive separation approach of the non-context sensitive part, decision part, and context-sensitive part, all consisting of unique sub-trees. Also the decision trees can be edited and maintained independently of each other, with their own logic. The major inconvenience of this approach is that it dramatically suffers from a proliferation of trees, even if unique trees, with no direct linking between them. Moreover, maintaining such series of trees in a repository becomes quite complex, especially for future algorithms for UI derivation, and the modelling becomes very different from the initial task model as noted in CTT.

The conclusion is that a better solution could be set up by combining the unique representation of the *monolithic approach* with decision tree and clear separation as discussed in the context-sensitive separation approach (context-sensitive part, non-context-sensitive part, and context decision tree) to produce a model that is closer to the initial CTT notation, while preserving the hierarchical structure of the model, as expressed in CTT.

The final solution would be:

- *A non-context-sensitive* part as typically modeled in a CTT task model;
- *A decision tree* represented by three CTT notation elements: the different contexts of use are linked sequentially as optional sub-tasks at the first level with the sequencing operator, then the different contextual conditions are expressed as sub-sub-tasks with choice operator (mutual exclusion), and finally the leaves of the decision trees are the roots for the context-sensitive part
- *A context-sensitive* part as a series of sub-trees as typically modeled in CTTE.

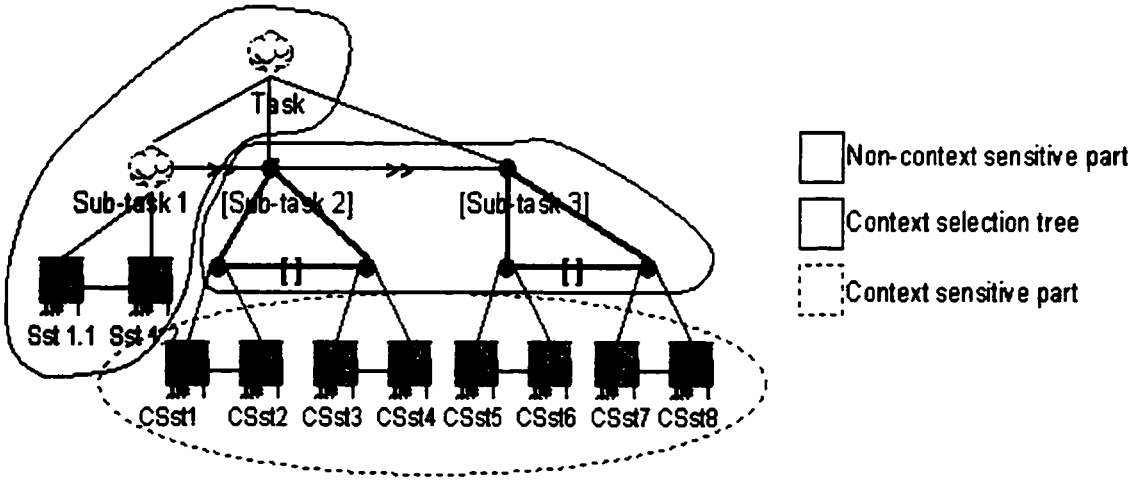

Figure 4-7 Final approach for modeling a context-sensitive task [24]

In order to make a distinction between the normal choice operator and the ones used for the decision tree, we decided to introduce a marked structured annotation in the tasks representing the nodes of the decision tree. Each such task is then augmented with the current value (e.g. low/moderate/high) of the contextual condition considered at that time.

We then choose to export the resulting CTT model into an XML-compliant file thanks to the facility provided by CTTE, an automatic tool for supporting the development of task models and their analysis. This XML file is then transformed into an XIML-compliant file using XSLT transformations. In particular, the arcs and nodes used for the decision tree are isolated in dedicated tags with the following structure:

<?xml version="1.0" encoding="ISO-8859-1" ? >

<!DOCTYPE csui PUBLIC "-//CSUI" "CSUI1_0.dtd">

<decision-tree>
<node> name
<node id> id </node_id>
<contextual-cond> condition </contextual-cond>
<context-value> value <context-value>
<task unit>... </task-unit>
<node> ... </node>
<node> ... </node> ...
</node>
</decision-tree>

This XIML file can then be exploited by UI derivation algorithms to progressively derive a generic UI, then a specific UI, and a code generation/interpretation.

The following table provides a comparison between some of the low-level and high-level frameworks from the supporting development of the context-sensitive point of view.

Language	Models	Output	Design Knowledge	Methodology	XML compliance	Supporting tool
UIML	Dialog Presentation Domain (parts)	Fully functional interface	None	None	Yes	Code generator
XIML	Task Domain User Dialog Presentation Platform Design	Fully functional interface	Design Guidelines Heuristics	Model-based approach	Yes	Rendering engine Code generator Editor

Table 4-3 Comparison of frameworks for context-sensitive UIs

While the UIML is ostensibly independent of the specific device and medium used for the presentation, it does not seem to take into account the research work carried out in the last decade on model-based approaches for user interfaces. In terms of supporting the particular use of the task model-based method we conclude that the language provides no notion of

task, it mainly aims to define an abstract structure and needs further improvement. XIML is a modeling language for specifying Task, Domain, User, Dialog, Presentation and Platform design models, generating in the same time a fully functional interface [31].

4.5 Framework for Applying the Presentation, Platform and Task models to the development of Mobile UIs

This framework exploits three UI models: *platform model*, *presentation model* and *task model* to facilitate development of a highly adaptive, context-sensitive, multi-platform UIs. The UI model developed with this approach serves to isolate those features that are common to the various contexts of use, and to specify how the UI should adjust when the context changes [28]. The approach places special emphasis on the connections (mappings) between the various models as these mappings will determine the interactive behavior of the UI. These mappings are then interpreted to produce a static or dynamic UI that is specially customized for the relevant device and context of use.

A *platform model* describes the various computer systems that may run a UI. A platform is modeled in terms of resources, which is turn, determine the way information is computed, transmitted, rendered and manipulated by users. This model includes information regarding the constraints placed on the UI by the platform (e.g. screen resolution of each device is represented declaratively). The platform model contains an element for each platform that is supported, and each element contains attributes describing features and constraints. The platform model may be exploited at design time and be used as a static entity. In this case, a set of user-interface can be generated: one for each platform that is desired. However, we prefer the dynamic exploitation of the platform model at run-time, so that it can be sensitive to changing conditions of use. For example, the platform model should recognize a sudden reduction in bandwidth, and the UI should respond appropriately.

A *presentation model* describes the visual appearance of the user interface. It is also possible to represent the amount of screen space required by each presentation structure. The presentation model includes information describing the hierarchy of windows and their

widgets (e.g., sliders, list boxes), stylistic choices, and the selection and placement of these widgets. Widgets are described in accordance with the traditional distinction between AIOs and CIOs [2]. Each widget is modeled abstractly as an AIO: an abstract interaction object, which is platform-neutral. Then each AIO is associated with several CIOs: concrete interaction objects, which are executable on a specific platform. CIOs inherit some of their behavior from AIOs, supplying also some additional parameters and are then mapped on to specific platforms. The distinction between the AIOs and CIOs allows our UI models to run on any computing platform, as long as the appropriate CIOs are present. It gives the layout of the interactors and their's allocation between windows. When the UI model is interpreted, the appropriate CIO is rendered automatically, depending on the platform.

If we want the system to generate the correct presentation structure given a set of user-defined constraints we need to consider besides AIOs and CIOs other 2 abstractions: the logical window (LW), a composite AIOs and PU (presentation unit). This represents a complete presentation environment required for carrying out a particular interactive task. Each PU can be decomposed into one or many LWs, which may be displayed on the screen simultaneously, alternatively, or in some combination thereof. The abstractions can be structured into a hierarchy that serves to expand modeling capabilities of a presentation model. We can use this hierarchy to construct an automated design tool that generates several platform-optimized presentation models from a starting presentation model that is platform independent.

The connections between the *presentation model* and the *platform model* describe how the constraints posed by the various platforms will influence the visual UI appearance (see Figure 4-8).

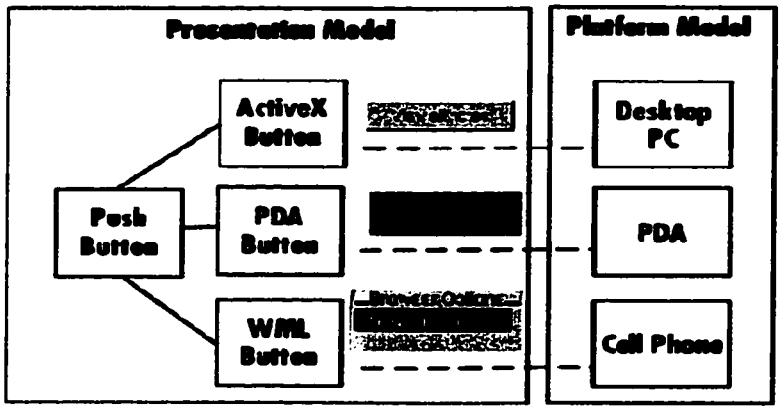

Figure 4-8 Mapping between Presentation and Platform Models [28]

There are three parameters that contribute to the amount of display size required by a UI design:

- *Individual interactors size.* The interactors could be shrinked while observing the usability constraints related to the AIO type. For example, the length of an edit box can be reduced to a minimum (e.g., 6 characters visible at the same time with horizontal scrolling) while its height cannot be decreased below the limit of the smallest font size legible (e.g., 8 pixels). Usability experiments have determined that the minimum size for an icon is roughly 8 by 6 pixels. An alternative to reducing the size dimensions of an interactor is to replace that interactor with a smaller alternative. A Boolean check-box typically requires less screen space than a pair of radio buttons. A WAP-enabled cellular phone can display only one or two interactors at a time, no matter how small they are.
- *Layout of interactors within a window*;
- *The allocation of interactors among several windows.*

We need to select the appropriate presentation structure to accommodate the constraint of the amount of screen-resolution afforded by the platform, in this way achieving the amount of flexibility necessary to support all mobile devices. The other constraints such as bandwidth usage, battery power consumption, number of display colors, and interaction capabilities can usually be accommodated through AIO selection. Limited interaction capabilities constraints are handled through *interactor selection.* The *interactor selection* can become a multidimensional optimization problem. Interactors can be parameterized to reduce

bandwidth usage or battery consumption–for example, a video player can reduce the frame rate or the number of colors.

Therefore we need a set of alternative presentation structures, which can be generated by the designer or with the help of the system, and then to create *mappings* between each *platform* and an appropriate *presentation* structure. This could be done dynamically using a mediator, which should determine the maximum usable screen resolution for the relevant device, and evaluate the amount of screen resolution required by each presentation structure alternative. It can then select the presentation structure that consumes an amount of screen resolution that falls just under the maximum. This solution is preferable because it accounts for the fact that the screen resolution of a device may change while it is in use. Moreover, it eases the integration of new devices into the platform model. Rather than forcing the user to explicitly specify the appropriate presentation structure for a new device, the user needs only to specify the amount of available screen space, and the mediator will find the correct mapping.

The *task model* should model features such as whether a task is optional, whether it may be repeated, and whether it enables another sub-task. We have to accommodate the fact that a device may be specially suited for a specific subset of the overall task model. Through the application of a task model, we can take advantage of this knowledge and optimize the UI for each device.

The designer should create *mappings* between *platforms* (or classes of platforms) and *tasks* (or sets of tasks). Additional mappings are then created between *task* elements and *presentation* layouts that are optimized for a given set of tasks (see Figure 4-9). We can assume these mappings are transitive; as a result, the appropriate presentation model is associated with each platform, based on mappings through the task model. There are several ways in which a presentation model can be optimized for the performance of a specific subset of tasks. Particularly important tasks should be represented by AIOs that are easily accessible. The automated generation of task-optimized presentation structures would be explored in the future.

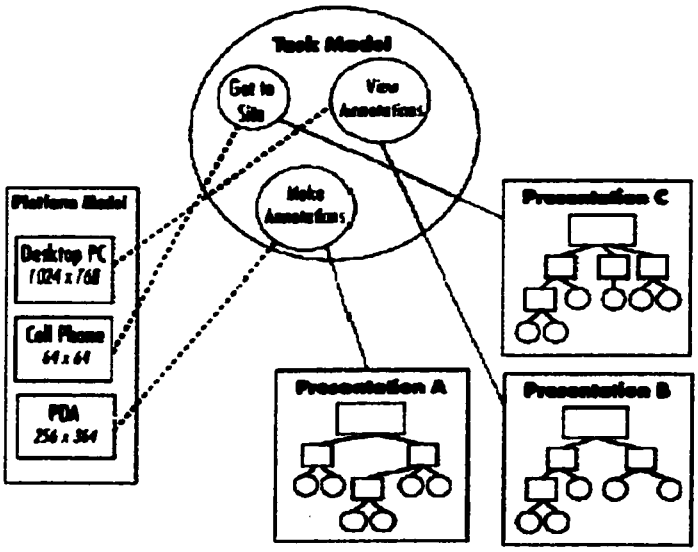

Figure 4-9 Mapping between Platform Model, Task Model and Presentation Models [28]

Much of the information found in the platform and task models can be expressed as UI constraints: e.g., screen size, resolution, colors, available interaction devices, task structure, and task parameters (e.g., frequency, motivation). It is therefore tempting to address the problem of generating alternative presentation structures as a constraint-satisfaction problem. In this case, there are two types of constraints: first, features that are common to the user-interface across platforms and contexts of use; second, platform-specific constraints such as screen resolution and available bandwidth.

As the UI guidelines are not always best represented as constraints, a trade-off between device constraints and some usability heuristics need to be considered when generating presentations structures (e.g. an AIO may be optimal in regard to usability but may consume too much space.

Thevenin *et al.* [29] are reusing these concepts for analyzing plastic UIs when a presentation and/or a dialog change according to any non-user variation such as the platform, the context.

4.6 Unified Framework for Development Process of Plastic UIs

Although the UIML and mechanism based on XML technology are technical attempts to address the issue of the plastic UIs development, they do not provide sufficient insight about how the UI adaptation process to different computational devices could happen.

This framework sustains the development of the plastic UIs when a presentation and/or a dialog change according to any non-user variation such as the platform, the context, and the interaction capabilities [29]. It uses the *domain concepts model*, describing the concepts the user manipulates in any context of use, and the *tasks model*, with improvements for accommodating variations of contexts of use. New models and heuristics are introduced to express contexts of use:

- *Platform model.* Describes the physical characteristics of the target platform. It should include the interactional devices available, the computational facilities (e.g. memory and processing power) and the communicational facilities (e.g. available bandwidth).
- *Environment model.* Describes the entities like objects, persons and events that are peripheral to the current task(s) but that may have an impact on the system and/or the user's behavior, either now or in the future. Specifies the context of use together with the *Platform model*;
- *Evolution model.* Specifies the change of state within a context as well as the conditions for entering and leaving a particular context.
- *Interactors model.* Describes "resources sensitive multimodal widgets" available for producing the concrete interface. Widgets may be functionally equivalent but may have very different costs (e.g., computational costs but also cognitive, conative, and physical costs for the user). They are the *initial models* in contrast to the transient model or intermediate model (e.g., the abstract and concrete user interfaces), inferred by the developer or system through the development process, and necessary for the production of the final executable UI.

All of the above models are referenced along the development process from the task specification to the running interactive system. The process is a combination of vertical reification and horizontal translation. Vertical reification covers the derivation process, from top level abstract models to run time implementation. Horizontal derivations, such as those performed between HTML and WML content descriptions, correspond to translations between models at the same level of reification. Reification and translation may be performed automatically from specifications or manually by human experts. Because automatic generation of UIs has not found wide acceptance in the past [2], this framework makes possible manual reifications and translations. Such operations are manual when the tools at hand cannot preserve the usability criteria set up for the particular system or when they simply do not exist. Figure 4-10 shows the process when applied to two distinct contexts: context1 and context2.

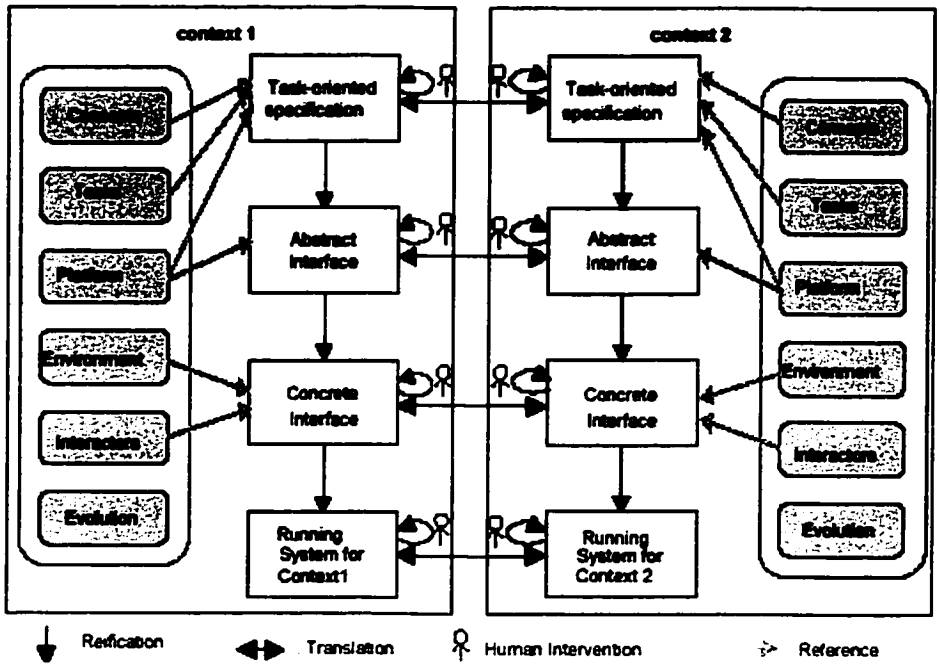

Figure 4-10 Reference development process for supporting plastic UIs [29]

ARTStudio (Adaptation by Reification and Translation) [29] is a tool designed to support this framework. The current implementation addresses the reification process only, and does not include the environment and the evolution models. According to the reference framework, the concepts and the task models serve the task-oriented specification which, in

turn, leads to the automatic generation of the abstract user interface. Then, the platform model and the interactors model come into play for the automatic generation of the concrete UI. ARTStudio is implemented in Java and uses the CLIPS rule-based language for generating the concrete user interface. Initial and transient models are saved as XML files. So far, final executable user interfaces are expressed in Java. As a result, the current version of ARTStudio does not support XML-based executables. Concepts are modeled as UML objects. The platform model is a UML description that captures the size and the depth of the screen, the programming language supported by the platform (e.g., Java). Thevenin *et al.* considers that ARTStudio provides a first step towards a systematic high quality development of plastic UIs [29].

4.7 XIML Framework

XIML (eXtensible Interface Markup Language) is a framework the representation and manipulation of interaction data which is the data that defines and relates all the relevant elements of a UI. The aim of XIML is to provide the software infrastructure needed to support web and application services for UIs. The XIML will be ultimately an XML-based mechanism designed to manage the complex interactions among users, applications, devices, and UIs [30, 31]. XIML has been under development for more than 2 years in the laboratories of this Palo Alto-based startup. It seeks to model exhaustively a UI, including user tasks, user profiles, and domain information. XIML will be the centerpiece of the development of a comprehensive UI infrastructure.

The XIML is an extensible XML specification language for multiple facets of multiple models in a model-based approach. An XIML specification can lead to both an interpretation at runtime and a code generation at design time. XIML offers a representational basis for UI transformations. It represents the critical elements of a user interface, along with their attributes and their relationships. In sum, it captures the structure and design of a user interface.

The main features of the language are:

- Represents the *concrete aspects* of a user interface (such as presentation and dialog) and also its *abstract aspects* (such as context).
- Bridges the gap between design and development of user interfaces by providing a single common representational framework for both processes.
- Provides a ready knowledge repository for runtime operations such as personalization, adaptation, context-aware and agent-based interaction.
- Provides mechanism for the distributed update of UI components.

Using XIML, the design and implementation of a user interface is conceived as a series of refinements from an abstract representation (of user context for example) to a concrete representation (of widgets and interaction techniques for example). The development process is visualized as the guided selection of appropriate transformations that effectively refine an abstract interface into a concrete one.

XIML is an organized collection of interface *elements* that are categorized into one or more major interface *components* (see Figure 4-11):

- *contextual* and *abstract*: *task*, *domain* and *user*;
- *implementational* and *concrete*: *dialog* and *presentation*.

The *task* component captures the business process and/or user tasks that the interface supports. The component defines a hierarchical decomposition of tasks and subtasks that also defines the expected *flow* among those tasks and the *attributes* of those tasks. The *domain* component is an organized collection of data objects viewed or manipulated by a user. The *user* component defines a hierarchy of users. A user in the hierarchy can represent a user group or an individual user. Attribute-value pairs define the characteristics of these users. The *presentation* component defines a hierarchy of interaction elements that comprise the concrete objects that communicate with users in an interface. It determines what widgets, interactors, and controls will be used to display each data item on each of the target devices. The *dialog* component defines a structured collection of elements that determine the interaction actions that are available to the users of an interface. For example, a "Click", a "Voice response", and a "Gesture" are all types of interaction actions. The dialog component also specifies the flow among the interaction actions that constitute the allowable *navigation*

of the user interface. This component is similar in nature to the Task component but it operates at the concrete levels as opposed to the abstract level of the Task component.

A *relation* in XIML is a definition or a statement that links any two or more XIML elements either within one component or across components. The set of relations in an XIML specification captures the *design knowledge* about a UI. The runtime manipulation of those relations constitutes the *operation* of the UI. In XIML, *attributes* are features or properties of elements that can be assigned a *value*.

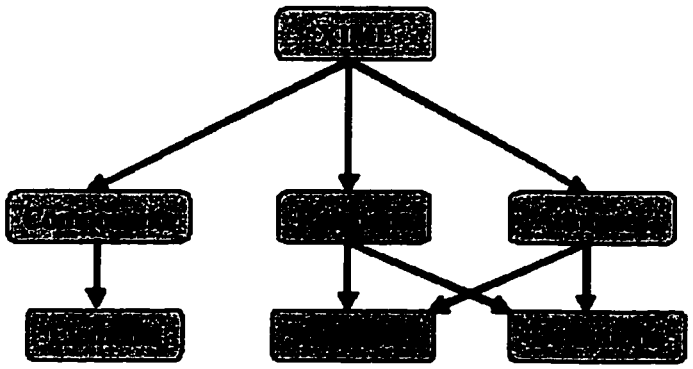

Figure 4-11 Representational Structure of XIML [31]

XIML supports multi-platform development of the UIs, by displaying a single interface definition on any number of target devices. In the XIML framework, the definition of the interface is the actual XIML specification and the rendering of the interface is left up to the target device to handle. This is an advantage over many model-based interface development systems that do not have this separation established clearly and therefore developers ended up mixing up interface logic with interface definition. In Figure 4-12 we see that by simply defining one presentation component per target device the entire specification can support multiple platforms. Besides the fact that a presentation component is needed for each target device, the framework saves development time and ensures consistency among UIs.

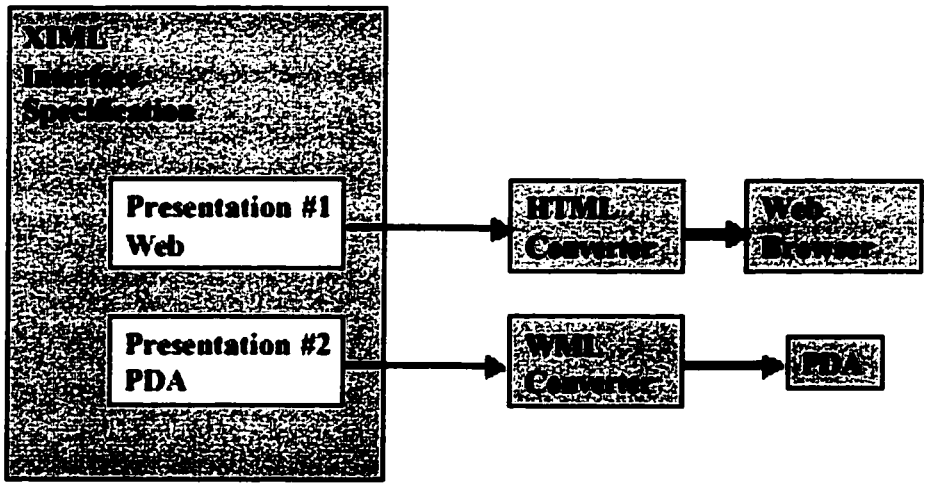

Figure 4-12 UI Deployment to different targets [31]

XIML has the necessary capabilities to provide support for the development of the frameworks described in the previous 3 sections. UIML, which is also a language for multi-platform development, has the following disadvantages in comparison with XIML language:

- It does not capture context data
- It is not intended to support knowledge-based system functions
- It does not target operation and evaluation functions
- It does not clearly separate the rendering of the interface from the definition of it.

4.8 Framework for User Preference Modeling for Adaptive User-Interfaces

Model-based UI development could be characterized as a process of creating mappings between elements in various model components. Rather than modeling the UI as a set of static structures and mappings, the UI should be modeled as a set of design preferences [32]. Preferences are frequently many-to-one or many-to-many relationships that elude conventional UI modeling, which has largely focused on one-to-one mappings. Preference relations allow for a more flexible and adaptable specification of the user-interface; rather than specifying exactly how the UI must appear, the designer can specify what would be preferred, and in which situation. This freedom is particularly important for user-interfaces that must run on heterogeneous devices, since the contexts of use vary and may even change at run-time. Preference modeling is also critical if interaction is to be dynamically customized for the user; *preference relations* can be used to show how the user-interface

should change in relation to the user model.

There are concrete preferences and abstract preferences (see Table 4-4). Concrete preferences specify their targets directly, although an ordered list of targets is permitted. Abstract preferences specify their targets indirectly, based on *criteria* that describe characteristics of either the targets themselves or some other *object*.

	Condition	Target	Criteria	Object
Binding	1	1	None	None
Simple Preference	Any number	1	None	None
Ordered Preference	Any number	Any number	None	None
Abstract Preference	Any number	More than 1	Criteria apply to targets; logical and preferential criteria are allowed	None
Design guideline	Any number	Any number	Criteria apply to object; only logical criteria are allowed	Any number

Table 4-4 Preference Relations [32]

The simple preference presented in Table 4-5 has the meaning that: "User U1 prefers presentation element P1 for representing domain model Dm1."

CONDITIONS	TARGETS	OBJECTS	CRITERIA
U1, Dm1	P1		

Table 4-5 Concrete Preference [32]

The preference presented in Table 4-6 has the meaning that: "For task model Tm2 and domain model Dm1, user U4 likes presentation elements that require few clicks, take up little screen real estate, and have lots of colors. The criteria of having lots of colors is most important, followed by the number of clicks. But if the amount of space occupied is too small, it won't be visible, so disallow it altogether."

Tm2, Dm1, U4	P1, P2,...Pn	Name Type Behavior Priority	Screen Space Preferential Minimize 25	
		Name Type Behavior Threshold	Screen Space Logical Greater Than 100	
		Name Type Behavior Priority	Clicks Preferential Minimize 50	
		Name Type Behavior Priority	Number of Colors Preferential Maximize 100	

Table 4-6 Abstract Preference [32]

The formalism for modeling preference relations has been incorporated into XIML as part of the Design Model Component. The design model consists of a list of preference elements. A preference element can have four child elements: conditions, targets, objects and criteria. Each of these elements contains a list of relation statements, which indicate a mapping to another element somewhere else in the UI specification. Relation statements are simple one-to-one mappings, with a reference to an element's ID and a semantic definition.

4.9 Framework for Context – Awareness Development

Efforts have already begun in developing frameworks that support building of context-aware applications. The *Context Toolkit* [33, 34] is an infrastructure to support the rapid development of context-aware services, assuming the sensing of perfect context. This framework's architecture enables the applications to obtain the context they require without knowledge about how the context was sensed.

The *Context Toolkit* (see Figure 4-13) consists of *context widgets* that implicitly sense context, *aggregators* that collect related context, *interpreters* that convert between context types and interpret the context, *applications* that use context and a *communications*

infrastructure that delivers context to these distributed components. It makes easy to add the use of context or implicit input to existing applications that don't use context.

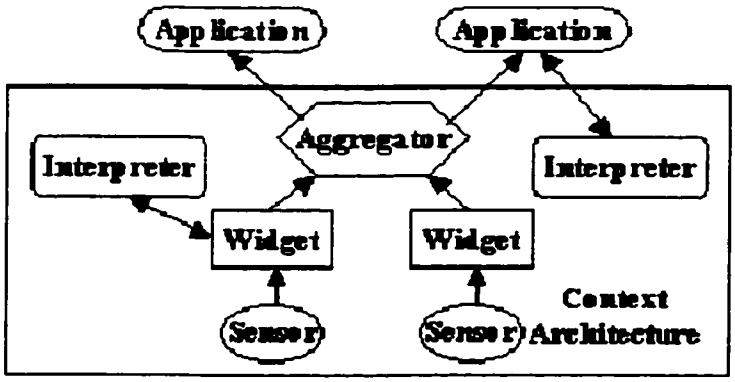

Figure 4-13 Context Toolkit Components [33]

Context widgets are software components that provide applications with access to context information while hiding the details of context sensing. In the same way GUI widgets insulate applications from some presentation concerns, context widgets insulate applications from context acquisition concerns. They have state and behavior. Context widgets mediate between the environment and the application in the same way graphical widgets mediate between the user and the application. Already widget libraries have been developed for sensing presence, identity and activity of people and things. A *context widget* encapsulate, and is responsible for acquiring, a certain type of context information like: activity, identity, location, time, and makes that information available to applications in a generic manner, regardless of how it is actually sensed. Widgets maintain a persistent record of all the context they sense. They allow other components to both poll and subscribe to the context information they maintain. Widgets are responsible for collecting information about the environment.

A *context interpreter* is used to abstract or interpret context. For example, a context widget may provide location context in the form of latitude and longitude, but an application may require the location in the form of a street name. A context interpreter may be used to provide this abstraction. A more complex interpreter may take context from many widgets in a conference room to infer that a meeting is taking place. A *context aggregator* is very similar to a widget, in that it supports the same set of features as a widget. The difference is that an

aggregator collects multiple pieces of context. In fact, it is responsible for the entire context about a particular entity (person, place, or object). Aggregation facilitates the access of context by applications that are interested in multiple pieces of context about a single entity.

This architecture will allow an entity to create dynamic relationships with other entities that share part of its context to identify, for example, all the people meeting in the same room.

The services of the Context Toolkit are: encapsulation of sensors, access to context data through a network API, abstraction of context data through interpreters, sharing of context data through a distributed infrastructure, storage of context data, including history, basic access control for privacy protection.

Context components are intended to be persistent, running 24 hours a day, 7 days a week. They are instantiated and executed independently of each other in separate threads and on separate computing devices. The Context Toolkit makes the distribution of the context architecture transparent to context-aware applications, handling all communications between applications and components. XML and HTTP are used for sending data, Java and C++ for widgets. There are components of the infrastructure executing on Windows CE, 95 and NT, Linux, Solaris, IRIX, and the Macintosh platforms, including mobile, wearable, and desktop machines. Currently this infrastructure supports only discrete context, and not continuos context such as video feed or GPS location information. In the future there is the need for supporting this type of context.

If we consider the possibility that the sensors could make a mistake, than we would have to deal with the context interpreters creating multiple ambiguous interpretations of the context. In the future we need to handle this situation by using mediation techniques for correcting the context. These mediation techniques could allow the users themselves to correct the context.

5. Chapter V - Conclusions and Future Development

In this thesis, we conducted an analysis of industry and academic research frameworks for mobile UI development. This analysis is a first step of an ongoing research project, which aims to define and develop a universal framework for mobile applications. The framework will overcome the current limitations of the existing frameworks while inheriting all their benefits. The following two tables summarizes some of the major results:

Criteria	HTML 3.2	WML	UIML	Java (MIDP)
XML-based	No	Yes	Yes	No
Learnability	Easy	Easy	Learning curve present	Sharp learning curve
Separation of content Versus Presentation	No	Moderately	Strongly	Strongly
Internationalization	No	Yes	Yes	Yes
Extensible	No	Moderately	Strongly	Strongly
Primary metaphor for UI construction	None	Deck of cards	None	MIDP: Screen

Table 5-1 Comparison of HTML 3.2 for handheld, WML, UIML and Java languages

Criteria	Web Clipping	WAP	UIML	J2ME
Delivery Mechanism Device profile specification	No	Not clearly specified	Not clearly specified	MIDP
Delivery Mechanism Gateway Required	Proxy Server	Yes	Optional	No
Delivery Mechanism Complexity of gateway	Translation from Web Clipping format into HTML 3.2 needed.	Encoding/Decoding capability required	Rendering capability needed.	No Encoding /Decoding required
Deployment	Wide	Wide	Few	Wide
Availability	No Standard	International Standard	Standard being developed	International Standard
Scalability in terms of types of devices	Palm Powered Handheld	Restricted to cellular phones and pagers, primarily	Available for any kind of device	Available for any kind of device
Security support available	Moderately	Moderately	Moderately	Highly
Data Transfer Multiple Modes	Limited	No	Yes	Yes
Data Transfer Wireless / Wireline	Both	Wireless, WAP Network	Both	WAP, TCP/IP
Data Access Local databases	No	No	No	No
Data Access Local processing	No	No	No	Yes
Data Interactivity Rich Display	Yes	No	Yes	Yes
Data Interactivity Flexible Input	Yes	No	Yes	Yes
Applications Type	Online catalogs/trading, travel	Stock quotes, weather forecasts and discount	Stock quotes, travel reservations,	Stock transactions, travel reservations, online purchase of

Criteria	Web Clipping	WAP	UIML	J2ME
	guides, real estate home finders	offers	online banking, news headlines, weather updates, and traffic information	concert tickets, news headlines, weather updates, and traffic information, games

Table 5-2 Comparison of Web Clipping, WAP, UIML and J2ME frameworks

We believe that future research on UI development framework should focus on automatic content filtering and reformatting. One possible avenue for such research would be the establishment of higher level patterns that could specify abstract information formats. These same patterns would also describe alternate implementation solutions for use on the different target platforms. Moreover, it is conceivable that these patterns be themselves implemented as a toolkit. Content developers would then only have to ensure that they conformed to the abstract specification defined by the pattern, and then could rely on pre-programmed real-time interpreters to refine the data for use on a particular target device.

We also highlighted that current UI implementers face uncertainty and risk in adopting new devices, because of the variety of UI languages, operating systems and OS APIs. To deploy a family of UIs for PDA, desktop, or phone, the development team needs to master PDA APIs (e.g., for the Palm), Java AWT or Swing, and WML. We argue once again that this situation impedes rapid adoption of new devices. Just as OSs were critical to shied multi-architecture developers in the PC industry, we argue that development of the device-independent UI markup languages is essential to speed adoption of new devices today. We claim that without some abstract description of the mobile UI, it is likely that the design and the development of UIs for mobile computing will be very time consuming, with significant chances for errors and failures.

Many research efforts have demonstrated that the model-based approach could provide a viable and effective new framework for developing user interfaces. Model-based interface

technology allowing UI code generation from models, is becoming more and more used for interactive systems development and in particular for mobile devices. Our central claim is that the use of abstract, platform-neutral models to describe the user-interface greatly facilitates the development of consistent, usable multi-platform user-interfaces for mobile devices.

It is clear that the research on user interface software tools has had enormous impact on the process of software development up to now. However, we expect to see further development of the UI software tools with respect to supporting the new design trends and methodologies that we described and proposed for mobile computing. It will be important for tools to provide facilities to manage data sharing and synchronization, especially since it must be assumed that the data will go out of date while disconnected and need to be updated when reconnected. Since data will be updated automatically by the environment and by other people, techniques for notification and awareness will be important. Furthermore, the tools will be needed to help present the information in an appropriate way for whatever device is being used. If the user has multiple devices in use at the same time, then the content and input might be spread over all the active devices.

With the rise of the use of the Internet and World-Wide Web, people will be more and more interacting with multiple devices at the same time so the devices will need to communicate and coordinate their activities. Decisions about what information is to be sent and shown to the user should be based on the importance and timeliness of the information and the capabilities of the current connection and devices. The tools will be needed to help present the information in an appropriate way for whatever device is being used. If the user has multiple devices in use at the same time, then the content and input might be spread over all the active devices. The capabilities that should be available to all applications will be important to be provided at a low level, which suggests that the supporting capabilities be included in the underlying operating system or toolkits. As the computing environment used to present data becomes distinct from the environment used to create or store information, interface systems will need to support information adaptation as a fundamental property of information delivery.

References

[1] Satyanarayanan, M. *Fundamental Challenges in Mobile Computing*. Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing, ACM Press, New York, Philadelphia, 20-28 May 1996.

[2] Szekely, P. *Retrospective and Challenges for Model-Based Interface Development*. Proceedings of 2nd Int. Workshop on Computer-Aided Design of User Interfaces CADUI'96, Namur, 5-7 Juin 1996.

[3] Puerta, A and Eisenstein, J. *Towards a general computational framework for model-based interface development systems*. Proceedings of 1999 International Conferences on Intelligent User Interfaces, ACM Press, Los Angeles, California, January 1999.

[4] *Palm OS User Interface Guidelines*. Available at: http://www.palmos.com/dev/

[5] *Java Look and Feel*, Available at: http://java.sun.com/products/jlf/ed2/guidelines.html

[6] Ramsay, M. and Nielsen, J., Nielsen Norman Group *WAP Usability Déjà Vu: 1994 All Over Again*. Report from a Field Study in London, December 2000. Available at: http://www.Nngroup.com/reports/wap

[7] *Web Clipping Developer's Guide*. Available at: http://www.palmos.com/dev/support/docs/webclipping/PalmWebClippingTOC.html

[8] Wireless Application Protocol: *Architecture Specification*, April 1998, *Wireless Application Environment Specification*, *Wireless Markup Language Specification*, *WMLScript Language Specification*. Available at http://www.wapforum.org.

[9] Zhang, H., Stroulia, E. Babel *Application Integration through XML specification of Rules*. ACM Press, 23rd International Conference on Software Engineering, Toronto, Canada, 2-19 May 2001.

[10] Wiecha, C., Szekely, P. *Transforming the UI for anyone, anywhere*. Proceedings of CHI'2001, Seattle, Washington, 31 March-5 April 2001.

[11] Abrams, M., Phanouriou, C., Batongbacal, A., Williams, S. and Shuster, J. *UIML: An Appliance-Independent XML User Interface Language*. Proceedings of the World Wide Web Conference, Toronto, Canada, May 1999.

[12] Abrams, M. and Phanouriou, C. *UIML: An XML Language for Building Device Independent User Interfaces*. XML'99, Philadelphia, Pennsylvania, December 1999.

[13] *Java 2 Platform, Micro Edition*, White Paper, Available at: http://java.sun.com/j2me/j2me-ds-0201.pdf

[14] *Applications for Mobile Information Devices*, White Paper, Sun Microsystyems, Inc. Available at: http://java.sun.com/products/midp/midpwp.pdf

[15] *MIDP APIs for Wireless Applications,* White Paper, Sun Microsystyems, Inc. Available at: http://java.sun.com/products/midp/midp-wirelessapps-wp.pdf

[16] Pawlan, M. *Introduction to Wireless Programming with the MID Profile*, September 2000. White Paper, Available at: http://wireless.java.sun.com/midp/articles/intro/

[17] Mahmoud, Q. *WAP for Java Developers: Develop WAP Applications with Servlets and JavaServer Pages*. White Paper, Available at: http://wireless.java.sun.com/enterprise/articles/wap/intro/

[18] Srikanth Raju *Device Programming: MIDP For Palm, Motorola I85s.* White Paper, Available at: http://www.sun.com/developers/evangcentral/totallytech/midp_prog.html

[19] *Motorola Lightweight Windowing Toolkit for the J2ME.* Available at: http://www.motorola.com/java

[20] Ali, M. F., Abrams, M., and Pérez-Quinõnez, M. *Multi-Platform User Interface Construction with Transformations using UIML,* position paper for Workshop "Transforming the UI for anyone anywhere" at CHI'2001, Seattle, Washington, 31 March-5 April 2001.

[21] Ali, M. F. and Abrams, M. *Simplifying Construction of Multi-Platform User Interfaces Using UIML*, UIML'2001Conference, Paris, France, March 2001.

[22] Phanouriou, C. UIML: An Appliance-Independent XML User Interface Language, Ph.D. Dissertation, Virginia Polytechnic Institute and State University, 2000.

[23] Muller, A., Forbig, P., Cap, C. *Model-Based User Interface Design Using Markup Concepts*. Proceedings of the Eight Workshop of Design, Specification and Verification of Interactive Systems, Glasgow, Scotland, UK, 13-15 June 2001.

[24] Pribeanu, C., Vanderdonckt, J., Limbourg, Q., Souchon, N. and Florins, M. *Task modeling for context sensitive user interfaces*. Proceedings of 8th International Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'2001 Glasgow, 13-15 Juin 2001.

[25] Paternò, F., Breedvelt-Schouten, I., deKonig, N. *Deriving Presentations from Task Models*. Proceedings EHCI'98, Heraklion, Crete, Greece, 14-18 September 1998. More details about *ConcurTaskTree* at: http://giove.cnuce.cnr.it/ctte.html

[26] Cheverst, K., Davies, N., Mitchell, K., Friday, A. and Efstratiou, C. *Developing a Context-aware Electronic Tourist Guide: Some Issues and Experiences.* Proceedings

of CHI'2000, Netherlands, April 2000. More details at:
http://www.comp.lancs.ac.uk/computing/research/mpg/most/guide.html

[27] Elwert, T., Schlungbaum, E. *Modelling and Generation of GUIs in the TADEUS approach.* In: P. Palanque, R. Bastide (Eds.): Designing, Specification, and Verification of Interactive Systems. Springer Verlag, 1995.

[28] Eisenstein, J., Vanderdonckt, J., Puerta, A. *Applying Model-Based Techniques to the Development of UIs for Mobile Computers.* Proceedings of ACM Conference On Intelligent User Interfaces IUI'2001, Francisco, California, 13-16 January 2002.

[29] Calvary, G., Coutaz, J., Thevenin, D. *A Unifying Reference Framework for the Development of Plastic User Interfaces.* Proceedings of IFIP WG 2.7 Conference on Engineering the User Interface EHCI'2001, Chapman & Hall, London, Toronto, Canada, 11-13 May 2001.

[30] Puerta, A., Eisenstein, J. *A Representational Basis for User Interface Transformations.* In: Wiecha, Ch., Szekely, P. (eds.): Proceedings of CHI'2001 Workshop "Transforming the UI for anyone, anywhere – Enabling an increased variety of users, devices, and tasks through interface transformations". Seattle, Washigton, 31 March-5 April 2001.

[31] Puerta, A., Eisenstein, J. *XIML: A Common Representation for Interaction Data.* IUI2002: ACM on Sixth International Conference on Intelligent User Interfaces. San Francisco, California, 13-16 January 2002. Available at: http://www.ximl.org/documents/XIMLBasicPaperES.pdf

[32] Eisenstein, J. *Modeling Preference for Adaptive User-Interfaces.* First International Conference on Universal Access in Human-Computer Interaction. Proceedings of HCI International 2001. New Orleans LA, 5-10 August 2001.

[33] Dey, A.K., Salber, D., Abowd, G.D *A Context-based infrastructure for smart environments.* Proceedings of the International Workshop on Managing Interactions in Smart Environments. MANSE'99, Trinity College, Dublin, Ireland, 13-14 December 1999.

[34] Dey, A.K., Abowd, G.D., Mankoff J. *Distributed mediation of imperfectly sensed context in aware environments.* GVU Technical report GIT-GVU-00-14, Georgia Tech Institute. Available at:
http://www.cc.gatech.edu/fce/errata/publications/uist-distributedMediation00.pdf

[35] Dey, A.K., Abowd, G.D. *Towards a Better Understanding of Context and Context-Awareness.* Proceedings of CHI'2000 Workshop on Context Awareness, Hague, 1-6 April 2000. Research report 2000-18e, GVU Center, Georgia University of Technology, Atlanta 2000.

Acronyms and Abbreviations

AIM	Abstract Interaction Model
AIO	Abstract Interaction Object
API	Application Programming Interface
AWT	Abstract Window Toolkit
CPU	Central Processing Unit
CLDC	Connected Limited Device Configuration
CTT	ConcurTaskTree
CTTE	ConcurTaskTree Editor
CIO	Concrete Interaction Object
CSS	Cascade Style Sheets
DOM	Document Object Model
DTD	Document Definition Type
GPS	Global Positioning System
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HDML	Handheld Device Markup Language Specification
IDE	Integrated Development Environment
J2ME	Java 2 Micro Edition
JVM	Java Virtual Machine
JSP	Java Server Pages
LWT	Lightweight Windowing Toolkit
MIDP	Mobile Information Device Profile
MVC	Model-View-Controller
OS	Operating System
PDA	Personal Digital Analyzer
PQA	Palm Query Application
QAB	Query Application Builder
SAX	Simple API for XML
SIM	Specific Interaction Model
UA	User Agent

UI	User Interface
UIO	User Interface Object
UIML	User Interface Markup Language
XSL	Extensible Stylesheet Language
XSLT	XML Stylesheet Language Transformation
XML	Extensible Markup Language
XIML	eXtensible Interface Markup Language
WAP	Wireless Application Protocol
WML	Wireless Markup Language

Annexes

Annex 1. Web Clipping and HTML Framework Example

Skeleton PQA Page [7]

```
<html>
<head>
        <title>Sample Page for PQA</title>
        <meta name="palmcomputingplatform" content="true">
        <meta name="palmlauncherrevision" content="1.0">
        <meta name="localicon" content="logo1.gif">
        <meta name="localicon" content="logo2.gif">
</head>
<body>
<!-- your text and links here -->
</body>
</html>
```

Skeleton Clipping Page [7]

```
<html>
<head>
        <title>Sample Clipping</title>
        <meta name="palmcomputingplatform" content="true">
        <meta name="historylisttext" content="Latest News - &date &time">
</head>
<body>
<!-- your text and links here -->
</body>
</html>
```

Card/Deck Task Shadowing [8]

```
<wml>
<template>
        <do type="options" name="do1" label="default">
                <prev/>
        </do>
</template>
<card id="first">
        <!-- deck-level do not shadowed. The card exposes the deck-level do as part of the
        current card -->
        <!-- rest of card -->
</card>
<card id="second">
        <!-- deck-level do is shadowed with noop. It is not exposed to the user -->
        <do type="options" name="do1">
                <noop/>
        </do>
        <!-- rest of card -->
</card>
<card id="third">
        <!-- deck-level do is shadowed. It is replaced by a card-level do -->
        <do type="options" name="do1" label="options">
                <go href="/options"/>
        </do>
        <!-- rest of card -->
</card>
</wml>
```

WML Deck Structure [8]

```
<wml>
<card>
        <p>
        <do type="accept">
                <go href="#card2"/>
        </do>
        Hello world! This is the first card...
        </p>
</card>
<card id="card2">
        <p>
        This is the second card. Goodbye.
        </p>
</card>
</wml>
<wml>
```

Fieldset Element [8]

```
<wml>
<card>
        <p>
        <do type="accept">
        <go href="/submit?f=$(fname)&amp;amp;l=$(lname)&amp;amp;s=$(sex)&amp;amp;a=$(age)"/>
        </do>
        <fieldset title="Name">
                First name:
                <input type="text" name="fname" maxlength="32"/><br/>
                Last name:
                <input type="text" name="lname" maxlength="32"/>
        </fieldset>
        <fieldset title="Info">
                <select name="sex">
                        <option value="F">Female</option>
                        <option value="M">Male</option>
                </select><br/>
                Age: <input type="text" name="age" format="*N"/>
        </fieldset>
        </p>
</card>
</wml>
```

WAP and JAVA Examples

MOBILEDATE.JAVA [17]

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MobileDate extends HttpServlet {

public void service (HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        // set content type for wireless data
        response.setContentType("text/vnd.wap.wml");

        // get the communication channel with the requesting client
        PrintWriter out = response.getWriter();

        // write the data
        out.println("<?xml version=\"1.0\"?>");
        out.println("<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML
        1.1//EN\"");
        out.println(" \"http://www.wapforum.org/DTD/wml_1.1.xml\">");
        out.println("<wml>");
        out.println("<card title=\"MobileDate\">");
        out.println(" <p align=\"center\">");
        out.println("Date and Time Service<br/>");
        out.println("Date is: "+ new java.util.Date());
        out.println("</p>");
        out.println("</card>");
        out.println("</wml>"); }}
```

MOBILEDATE.JSP [17]

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<% response.setContentType("text/vnd.wap.wml");
out.println("<wml>");
out.println("<card title=\"MobileDate\">");
out.println(" <p align=\"center\">");
out.println("Date and Time Service<br/>");
out.println("Date is: "+ new java.util.Date());
out.println("</p>");
out.println("</card>");
out.println("</wml>"); %>
```

Annex 3. XML Framework Example

BOOKTEST.XML

```
<?xml version="1.0" ?>
<book><title>Content Transformation Example</title>
        <author>The Author</author>
        <publisher>Wrox</publisher>
        <tableOfContents showPageNumbers="yes">
                <tocEntry>Printing</tocEntry>
                <tocEntry>Cut&Paste</tocEntry>
                <tocEntry>Drag&Drop></tocEntry>
        </tableOfContents></book>
```

BOOKTRAN.XSL

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl=http://www.w3.org/1999/XSL/Transforms version="1.0">
<xsl:template match="book">
<html><body>
        <center><h1><xsl:value-of select="title"/><h1></center>
        <h4><xsl:value-of select="author"/></h4>
        <h3 Table Of Contents </h3>
        <ul><xsl: for-each select=" tableOfContents/ tocEntry">
        </xsl: for-each></ul>
</body></html></html>
</xsl:template>
</xsl:stylesheet>
```

Performing an XSL Transformation on an XML file

```
import org.apache.xalan.xslt.*;
public class XSLTest {
        public static void main *String args[]) throws org.xml.sax.SAXException {
                XSLTProcessor processor = XSLTProcessorFactory.getprocessor();
                XSLTInputSource xmlFile = new XSLTInputSource("booktest.xml");
                XSLTInputSource xslFile = new XSLTInputSource("booktran.xsl");
                XSLTResultTarget outputFile = new XSLTResultTarget ("booktest.html");
                Processor.process(xmlFile, xslFile, outputFile); }}
```

The generated HTML content: BOOKTEST.HTML

```
<html><body>
<center><h1> Content Transformation Example </h1></center>
<h4> The Author </h4>
<h3>Table Of Contents</h3>
<ul><li>Printing</li>
        <li>Cut&Paste </li>
        <li> Drag&Drop </li></ul>
        </body>
</html>
```

Annex 4. UIML Framework Example

An Online Banking Application

- **External use of Java Renderer**

JAVASAMPLE.UIML

```
<?xml version="1.0"?>
<!DOCTYPE uiml PUBLIC "-//UIT//DTD UIML 2.0 Draft//EN" "UIML2_0g.dtd">
<uiml>
<head><meta name="Description" content="Creating a Java UI for Online Banking"/></head>
<interface id="Swing"><!--decribes the Swing UI elements, their attributes and their behavior -->
<structure id="s1">
    <part id="Topwindow" class="JFrame"> <...>
    <part id="Balance" class="JButton"/> <part id="Withdraw" class="JButton"/>
    <part id="EndPanel" class="JPanel"><part id="Status" class="JLabel"/></part>
</structure>
<style id="Layout1"> <!--Version 1 vertical arrangement of the buttons-->
<property part-name="Topwindow" name="title">Online Banking Simulation Layout 1</property>
<property part-name="Topwindow" name="size">600,330</property>
<property part-name="Balance" name="text"><reference constant-name="BalanceC"/> </property>
<property part-name="Balance" name="bounds">60,125,200,40</property>
<property part-name="Withdraw" name="text"><reference constant-name="WithDrawC"/>
<property part-name="Withdraw" name="bounds">60,180,200,40</property>
<property part-name="EndPanel" name="background">046246</property>
<property part-name="EndPanel" name="layout">java.awt.FlowLayout</property>
<property part-name="EndPanel" name="bounds">10,250,590,40</property>
<property part-name="Status" name="text"><reference constant-name="StatusC"/></property>
</style>
<style id="Layout2"> <!--Version 2 horizontal arrangement of the buttons--><...><style>
<content id="English">
<constant id="BalanceC" value="Show Balance"/>
<constant id="WithDrawC" value="Withdraw"/>
<constant id="HeadlineC" value="Please select one of the following options"/>
<constant id="BalanceSuccess" value="Status: Please wait to calculate the Balance"/>
<constant id="WithdrawSuccess" value="Status: Please wait until the system is ready'">
<constant id="BalanceNoSuccess" value="Status: The balance cannot be calculated"/>
<constant id="WithdrawNoSuccess" value="Status: The system is not able to proceed"/>
```

```
<constant id="StatusC" value="Status: The system is back to normal"/></content>
<content id="French"> <!--defining all expressions in French -->
    <constant id="BalanceC" value="Montrer Balance"/>
    <constant id="WithDrawC" value="Retirer"/>
    <constant id="HeadlineC" value="Sélectionner une des options suivantes"/>
    <constant id="BalanceSuccess" value="Statut: Attendez pour calculer la Balance"/>
    <constant id="WithdrawSuccess" value="Statut: Attendez pour pouvez Retirer"/>
    <constant id="BalanceNoSuccess" value="Statut: On peut pas calculer la Balance"/>
    <constant id="WithdrawNoSuccess" value="Statut: On peut pas Retirer"/>
    <constant id="StatusC" value="Statut: Le system est prêt a fonctionner"/></content>
<behavior id="Success">
<rule> <!--listens if BalanceButton is pressed and changes the status line-->
<condition><event class="actionPerformed" part-name="Balance"/></condition>
<action><property part-name="Status" name="text" ><reference constant-
name="BalanceSuccess"/></property></action></rule>
<rule> <!--listens if WithdrawButton is pressed and changes the status line-->
<condition><event class="actionPerformed" part-name="Withdraw"/></condition>
<action><property part-name="Status" name="text"><reference constant-
name="WithdrawSuccess"/></property></action></rule></behavior>
<behavior id="NoSuccess">
<rule><condition><event class="actionPerformed" part-name="Balance"/></condition>
<action><property part-name="Status" name="text"><reference constant-
name="BalanceNoSuccess"/></property></action></rule>
<rule><condition><event class="actionPerformed" part-name="Withdraw"/></condition>
<action><property part-name="Status" name="text" ><reference constant-
name="WithdrawNoSuccess"/></property>
</action></rule></behavior>
</interface> <!--End of Swing Version-->
<interface id="NoSwing"><!--decribes the elements of the AWT UI, attributes and their behavior -->
<structure id="s1">
<part id="Topwindow" class="Frame"> <!-- root element --><...></structure>
<style id="Layout1"> <!--Version 1 vertical arrangement of the buttons--><...></style>
<style id="Layout2"> <!--Version 2 horizontal arrangement of the buttons-->
<property part-name="Topwindow" name="title">Online Banking Simulation </property>
<property part-name="Topwindow" name="background">046246</property><...>
<property part-name="EndPanel" name="layout">java.awt.FlowLayout</property>
<property part-name="EndPanel" name="bounds">10,250,590,40</property>
<property part-name="Status" name="text"><reference constant-name="StatusC"/></property>
</style>
```

```
<content id="English"> <!--defining all expressions in English --><...></content>
<content id="French"> <!--defining all expressions in French --><...></content>
<behavior id="Success"><...> </behavior>
<behavior id="NoSuccess"><...></behavior></interface>
<peers>
  <!--includes the vacabulary of the Java AWT and Java Swing -->
  <presentation base="Java_1.3_Harmonia_1.0" source="Java_1.3_Harmonia_1.0.uiml#vocab"/>
</peers></uiml>
```

- **Internal use of the Java Renderer**

JAVAINCLUDE1.JAVA

```
//Displays an UIML designed Window
import com.harmonia.renderer.Renderer;  //include the Rendering engine
public class javainclude1 {
        static String name = "myjavaincludeSample1.uiml";
        public static void main(String[] args)  {
                Renderer r = new Renderer();  //instantiate the renderer
                boolean ok = r.renderUIML(name); //run the renderer}}
```

MYJAVAINCLUDESAMPLE1.UIML

```
<?xml version="1.0"?>
<!DOCTYPE uiml PUBLIC "-//UIT//DTD UIML 2.0 Draft//EN" "UIML2_0g.dtd">
<!-- the same code as in javasample.uiml, but only the Swing part -->
<uiml><head><meta name="Description" content="Creating a UI for Online Banking"/></head>
<interface id="NoSwing">
<structure id="s1">
        <part id="Topwindow" class="Frame"><part id="Headline" class="Label"/>
        <part id="Balance" class="Button"/><part id="Withdraw" class="Button"/>
        <part id="EndPanel" class="Panel"><part id="Status" class="Label"/>
<tructure>
<style id="Layout1"><...>
<property part-name="EndPanel" name="layout">java.awt.FlowLayout</property><...></style>
<style id="Layout2"><...></style>
<content id="English"><...></content>
<content id="French"><...></content>
<behavior id="Success"><...></behavior>
<behavior id="NoSuccess"><...></behavior>
```

```
</interface>
<interface id="Swing">
  <structure id="s1">
      <part id="Topwindow" class="JFrame"><part id="Title" class="JLabel"/>
      <part id="Headline" class="JLabel"/><part id="Balance" class="JButton"/>
      <part id="Withdraw" class="JButton"/><part id="EndPanel" class="JPanel">
      <part id="Status" class="JLabel"/></part></part>
</structure>
<style id="Layout1"><...>
<property part-name="EndPanel" name="layout">java.awt.FlowLayout</property><...></style>
<style id="Layout2"><...></style>
<content id="English"><...></content>
<content id="French"><...></content>
<behavior id="Success">
<rule><condition><event class="actionPerformed" part-name="Balance"/></condition>
<action><property part-name="Status" name="text"><reference constant-
name="BalanceSuccess"/></property>
</action></rule>
<rule><condition><event class="actionPerformed" part-name="Withdraw"/></condition>
<action><property part-name="Status" name="text"><reference constant-
      name="WithdrawSuccess"/></property>
</action></rule></behavior>
<behavior id="NoSuccess">
<rule><condition><event class="actionPerformed" part-name="Balance"/></condition>
<action><property part-name="Status" name="text"><reference constant-
name="BalanceNoSuccess"/></property></action></rule>
<rule><condition><event class="actionPerformed" part-name="Withdraw"/></condition>
<action><property part-name="Status" name="text"
        ><reference constant-name="WithdrawNoSuccess"/></property>
</action></rule></behavior></interface>
<peers>
    <presentation base="Java_1.3_Harmonia_1.0" source="Java_1.3_Harmonia_1.0.uiml#vocab"/>
</peers></uiml>
```

JAVAINCLUDE2.JAVA

```
import com.harmonia.renderer.Renderer;  //include the Rendering engine
import javax.swing.*;
import java.awt.*;
```

```
public class javainclude2 {
        static String name = "myjavaincludeSample2.uiml";
        public static void main(String[] args)  {
                JFrame f = new JFrame("Create through Java Code Online Banking Main Window");
                f.getContentPane().setLayout(null);
                f.setSize(new Dimension(700,400));
                JPanel p = new JPanel();
                f.getContentPane().add(p);
                Renderer r = new Renderer();
                r.setExternalContainer(p);
                boolean ok = r.renderUIML(name);
                f.addWindowListener(new java.awt.event.WindowAdapter() {
                public void windowClosing(java.awt.event.WindowEvent we) {
                        System.exit(0); }       });
     f.show(); }
```

MYJAVAINCLUDESAMPLE2.UIML

```
<?xml version="1.0"?>
<!DOCTYPE uiml PUBLIC "-//UIT//DTD UIML 2.0 Draft//EN" "UIML2_0g.dtd">
<!-- modified version of javasample.uiml to run in a pre-created JavaFrame-->
<uiml><head><meta name="Description" content="Creating a UI for Online Banking"/></head>
<interface id="Swing">
  <structure id="s1"> <!--decribes the layout of the elements -->
       <part id="MainPanel" class="JPanel"><part id="Title" class="JLabel"/>
       <part id="Headline" class="JLabel"/><part id="Balance" class="JButton"/>
       <part id="Withdraw" class="JButton"/><part id="EndPanel" class="JPanel">
      <part id="Status" class="JLabel"/></part></part></structure>
  <!--properties for Main Window are set in the Java program-->
  <style id="Layout1">
   <!--set properties for Main Panel, instead of MainWindow-->
<property part-name="MainPanel" name="background">046246</property>
<property part-name="MainPanel" name="layout">null</property><...>
<property part-name="MainPanel" name="size">600,330</property>
<property part-name="Headline" name="text"><reference constant-name="HeadlineC"/></property>
<property part-name="Balance" name="text"><reference constant-name="BalanceC"/> </property>
<property part-name="Balance" name="bounds">60,125,200,40</property><...>
<property part-name="EndPanel" name="layout">java.awt.FlowLayout</property>
<property part-name="EndPanel" name="bounds">10,250,590,40</property>
```

```
<property part-name="Status" name="text"><reference constant-name="StatusC"/></property>
</style>
  <content id="English">
   <constant id="BalanceC" value="Show Balance"/>
   <constant id="WithDrawC" value="Withdraw"/>
   <constant id="HeadlineC" value="Please select one of the following options"/>
   <constant id="BalanceSuccess" value="Status: Please wait to calculate the Balance"/>
   <constant id="WithdrawSuccess" value="Status: Please wait until the system is ready"/>
   <constant id="BalanceNoSuccess" value="Status: The balance cannot be calculated"/>
   <constant id="WithdrawNoSuccess" value="Status: The system is not able to proceed"/>
   <constant id="StatusC" value="Status: The system is back to normal"/></content>
  <content id="French"><...></content>
  <!--does not effect the UI, as behavior has to be caught now by the Java Program-->
  <behavior id="Success">
   <rule><condition><event class="actionPerformed" part-name="Balance"/></condition>
    <action><property part-name="Status" name="text"
          ><reference constant-name="BalanceSuccess"/></property>
    </action></rule>
   <rule><condition><event class="actionPerformed" part-name="Withdraw"/></condition>
    <action><property part-name="Status" name="text"
          ><reference constant-name="WithdrawSuccess"/></property>
    </action></rule></behavior>
  <behavior id="NoSuccess">
   <rule><condition><event class="actionPerformed" part-name="Balance"/></condition>
    <action><property part-name="Status" name="text"
          ><reference constant-name="BalanceNoSuccess"/></property>
    </action></rule>
   <rule><condition><event class="actionPerformed" part-name="Withdraw"/></condition>
    <action><property part-name="Status" name="text"
          ><reference constant-name="WithdrawNoSuccess"/></property>
    </action></rule></behavior></interface>
 <peers>
  <presentation base="Java_1.3_Harmonia_1.0" source="Java_1.3_Harmonia_1.0.uiml#vocab"/>
 </peers></uiml>
```

- **Using HTML Renderer**

HTMLSAMPLE.UIML

```
<?xml version="1.0"?><!--Shorthand Notation, to reduce the amount of code-->
```

```
<uiml>
<head><meta name="Description" content="Creating a HTML UI for Online Banking"/></head>
<interface >
<structure id="Layout1"> <!--Version 1 vertical arrangement of the buttons-->
    <Html> <!--TopElement has to be HTML-->
     <Head id="Header"/>
     <Body id="Page">
      <H1 id="Headline"/><H3 id="Options"/>
      <Form><Button id="Balance"/><Button id="Withdraw"/></Form>
      <H3 id="Status"/>
     </Body></Html></structure>
<structure id="Layout2"> <!--Version 2 horizontal arrangement of the buttons-->
    <Html>
     <Head id="Header"/>
     <Body id="Page">
      <H1 id="Headline"/><H3 id="Options"/>
      <Form><Button id="Balance"/><Button id="Withdraw"/></Form>
      <H3 id="Status"/>
     </Body></Html></structure>
<style>
<property part-name="Page" name="bgcolor">046246</property>
<!--set the HTML Head Content, which is stored in the Variable HeaderC-->
<property part-name="Header" name="content"><reference constant-name="HeaderC"/></property>
<property part-name="Headline" name="content">Online Banking</property>
<property part-name="Headline" name="align">CENTER</property>
<property part-name="Options" name="content"><reference constant-
name="HeadlineC"/></property>
<property part-name="Options" name="align">LEFT</property>
<property part-name="Balance" name="value"><reference constant-name="BalanceC"/></property>
<property part-name="Withdraw" name="value"><reference constant-
name="WithDrawC"/></property>
<property part-name="Status" name="content"><reference constant-name="StatusC"/></property>
<property part-name="Status" name="align">CENTER</property></style>
<content id="English"> <!--English version of the expressions-->
    <!--set Labels for the Balance, Withdrawbutton and the Headline and Statustext-->
    <constant id="BalanceC" value="  Show Balance  "/>
    <constant id="WithDrawC" value="       Withdraw    "/>
    <constant id="HeadlineC" value="Please select one of the following options"/>
    <constant id="StatusC" value="Status: The system is back to normal"/>
    <!--the different status messages are used in Javascript mehtods, therefore we declare them in the
      HTML Header-->
    <!--HeaderC contains the entire HTML Header-->
        <constant id="HeaderC" value='&lt;title>Online Banking Simulation&lt;/title> &lt;script
        Language="JavaScript">
       var BalanceSuccess = "Status: Please wait to calculate the Balance";
       var WithdrawSuccess ="Status: Please wait until the system is ready for Withdraw";
       var BalanceNoSuccess ="Status: The balance cannot be calculated";
       var WithdrawNoSuccess ="Status: The system is not able to proceed the Withdraw";
       var m = null;
       &lt;/script>'/>
<!-- m contains the current status, and is assigned in the Javascript methods, coded in peers part-->
```

```
</content>
<content id="French"> <!--French version of the expressions--><...></content>
<behavior id="Success"> <!--Success case-->
  <rule> <!--listenes if BalanceButton is pressed and calls corresponding Javascript methods-->
   <condition><event class="OnClick" part-name="Balance"/></condition>
   <action>
    <call name="form.balancesuccess"/> <!--sets text for the Statuswindow-->
    <call name="form.status"/> <!--displays status window-->
   </action></rule>
  <rule> <!--listenes if WithdrawButton is pressed and calls corresponding Javascript methods-->
   <condition><event class="OnClick" part-name="Withdraw"/></condition>
   <action>
    <call name="form.withdrawsuccess"/> <!--sets text for the Statuswindow-->
    <call name="form.status"/> <!--displays status window-->
   </action></rule></behavior>
<behavior id="NoSuccess"> <!--Success case-->
  <rule><condition><event class="OnClick" part-name="Balance"/></condition>
   <action>
    <call name="form.balancenosuccess"/> <!--sets text for the Statuswindow-->
    <call name="form.status"/> <!--displays status window-->
   </action></rule>
  <rule><condition><event class="OnClick" part-name="Withdraw"/></condition>
   <action>
    <call name="form.withdrawnosuccess"/> <!--sets text for the Statuswindow-->
    <call name="form.status"/> <!--displays status window-->
   </action></rule></behavior></interface>
<peers>
  <!--HTML 3.2 vocabulary-->
  <presentation how="replace" source="HTML_3.2_Harmonia_1.0.uiml#vocab"
base="HTML_3.2_Harmonia_1.0"/>
  <!--extention for Java-Script-->
  <logic>
   <d-component id="form">
    <!--displays status window-->
    <d-method id="status" maps-to="displaystatus">
     <script type="text/javascript">
      function displaystatus () {        alert(m);          }
      function setbalancesuccess () {        m = BalanceSuccess;          }
      function setwithdrawsuccess () {        m = WithdrawSuccess;          }
      function setbalancenosuccess () {        m = BalanceNoSuccess;          }
      function setwithdrawnosuccess () {        m = WithdrawNoSuccess;          }
     </script></d-method>
    <d-method id="balancesuccess" maps-to="setbalancesuccess">
     <script type="text/javascript"></script></d-method>
    <d-method id="withdrawsuccess" maps-to="setwithdrawsuccess">
     <script type="text/javascript"></script></d-method>
    <d-method id="balancenosuccess" maps-to="setbalancenosuccess">
     <script type="text/javascript"></script></d-method>
    <d-method id="withdrawnosuccess" maps-to="setwithdrawnosuccess">
     <script type="text/javascript"></script></d-method>
   </d-component></logic></peers></uiml>
```

- **Using HTML Renderer**

WMLSAMPLE.UIML

```
<?xml version="1.0"?>
<!DOCTYPE uiml PUBLIC"-//Harmonia//DTD UIML 2.0 Draft//EN" "UIML2_0g.dtd">
<uiml>
<head><meta name="Description" content="Creating a WML UI for Online Banking"/></head>
 <interface>
  <structure>
   <!-- Mainpage contains of Maincard and StatusB and StatusW-->
   <part id="Mainpage" class="Wml">
    <!--Maincard-->
    <part id="Maincard" class="Card">
     <part id="Headpara" class="Paragraph">
      <part id="bold1" class="Bold">
       <part id="Headline" class="Text"/></part></part>
     <part id="free1" class="Paragraph"/>
     <part id="Middlepara" class="Paragraph">
      <part id="option" class="Text"/></part>
     <part id="free2" class="Paragraph"/>
     <part id="Optionpara" class="Paragraph">
      <part id="slctMenu" class="Menu">
       <part id="balance" class="Option">
          <part id="balancetxt" class="Text"/></part>
       <part id="withdraw" class="Option">
          <part id="withdrawtxt" class="Text"/>
       </part></part></part></part>

   <!--StatusB, displayed after Balance was choosen-->
   <part id="StatusB" class="Card">
    <part id="titleparaB" class="Paragraph">
     <part id="bold2" class="Bold">
      <part id="titleB" class="Text"/>
     </part> <!--End of bold2 -->
    </part> <!-- End of titleparaB-->
    <part id="free3" class="Paragraph"/>
    <part id="statparaB" class="Paragraph">
     <part id="statB" class="Text"/>
    </part> <!--End of statparaB --></part> <!--End of StatusB-->
   <!--StatusW, displayed after Withdraw was choosen-->
   <part id="StatusW" class="Card">
    <part id="titleparaW" class="Paragraph">
     <part id="bold3" class="Bold">
      <part id="titleW" class="Text"/></part></part>
    <part id="free4" class="Paragraph"/>
    <part id="statparaW" class="Paragraph">
     <part id="statW" class="Text"/></part></part></part></structure>
<style>
   <!--placeholder-->
   <property part-name="free1" name="content">&lt;p>&lt;br/>&lt;/p></property>
```

```
    <property part-name="free2" name="content">&lt;p&lt;br/>&lt;/p></property>
    <property part-name="free3" name="content">&lt;p&lt;br/>&lt;/p></property>
    <property part-name="free4" name="content">&lt;p&lt;br/>&lt;/p></property>
    <property part-name="Headpara" name="align">center</property>
    <property part-name="Headline" name="content">Online-Banking</property>
    <property part-name="option" name="content"><reference constant-
name="HeadlineC"/></property>
    <property part-name="balancetxt" name="content"><reference constant-
name="BalanceC"/></property>
    <property part-name="withdrawtxt" name="content"><reference constant-
name="WithDrawC"/></property>
    <property part-name="titleparaB" name="align">center</property>
    <property part-name="titleB" name="content">Status</property>
    <property part-name="statB" name="content"><reference constant-
name="BalanceSuccess"/></property>
    <property part-name="titleparaW" name="align">center</property>
    <property part-name="titleW" name="content">Status</property>
    <property part-name="statW" name="content"><reference constant-
name="WithdrawSuccess"/></property></style>
<content id="English"><...></content>
<content id="French"><...></content>
<behavior>
    <rule> <!-- if balance is choosen, display card for balance status -->
     <condition><event class="onpick" part-name="balance"/></condition>
       <action><call name="card.go"><param name="href">#statusB</param></call>S</action>
    </rule>
    <rule> <!-- if withdraw is choosen, display card for withdraw status -->
     <condition><event class="onpick" part-name="withdraw"/></condition>
       <action><call name="card.go"><param name="href">#statusW</param></call>S</action>
    </rule></behavior></interface>
<peers>
 <!--including the vocabulary for WML -->
 <presentation base="WML_1.3_Harmonia_0.3" source="WML_1.3_Harmonia_0.3.uiml#vocab"/>
     <logic>
       <d-component id="card" maps-to="card">
         <d-method id="go" maps-to="go">
           <d-param id="href" type="String"/>
           <d-param id="method" type="String"/>
           <d-param id="enctype" type="String"/>
         </d-method>
         <d-method id="prev" maps-to="prev"/>
         <d-method id="refresh" maps-to="refresh"/>
         <d-method id="noop" maps-to="noop"/>
       </d-component>
     </logic>
  </peers>
</uiml>
```

Annex 5. J2ME Framework Example

HelloMIDlet [16]

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class HelloMIDlet extends MIDlet
                    implements CommandListener {
private Command exitCommand;
private Display display;
private TextBox t = null;

public HelloMIDlet() {
        display = Display.getDisplay(this);
        exitCommand = new Command("Exit", Command.EXIT, 2);
        t = new TextBox("Hello MIDlet", "Test string", 256, 0);
        t.addCommand(exitCommand);
        t.setCommandListener(this);
}

public void startApp() {display.setCurrent(t);}

public void pauseApp() {}

public void destroyApp(boolean unconditional) {}

public void commandAction(Command c, Displayable s) {
if (c == exitCommand) {
        destroyApp(false);
        notifyDestroyed();
}
```