

HCI Designers and Engineers: It is possible to work together?

Jorge Belenguer, Jose Parra, Ismael Torres, & Pedro J. Molina.

CARE Technologies S.A., Pda. Madrigueres 44, 03700, Denia, SPAIN

{[jbelenguer](mailto:jbelenguer@care-t.com)|[jparra](mailto:jparra@care-t.com)|[itorres](mailto:itorres@care-t.com)|[pjmolina](mailto:pjmolina@care-t.com)}@care-t.com

Abstract: This position paper states the point of view of the authors about the problems and some possible solutions about integrating and making effective the work of Software Engineers and HCI professionals. These two groups of people use quite different jargon and tools to build User Interfaces. Try to bridging the gap and improve the communication between these two groups is essential to obtain better products in terms of efficiency (SE) and usability (HCI).

Keywords: Software Engineering, HCI, heterogeneous development teams.

1 Introduction

It is quite clear that the development of good systems requires the collaboration of different professionals: software engineers and programmers (SE) from one side and HCI professionals (including interaction designers, usability experts, graphic designers, user experience experts, etc.) from the other side are obliged to reach an agreement to satisfy the user.

Different groups of people use different jargon, methods, and techniques to develop UIs. The main problems arise from such differences and produces communication problems among the staff.

2 Background of the authors

All the authors of this position paper work for a Research & Development company interested in the automatic production of business applications (<http://www.care-t.com>). Therefore, we should warn in advance that our approach could be understood in some way a bit Software Engineering biased.

However, we want to describe our ideas in this workshop to stimulate discussion and obtain feedback from the research community, especially from approaches more HCI oriented.

2.1 Software Engineering approach

The approach we have taken is to use Conceptual Modelling to produce abstract and precise specification describing the system to be built. Such a system is specified in terms of structure, behaviour,

functionality and abstract user interface. Design details and platform selection are explicitly not collected in this phase to focus on *what it is needed* instead of *how it is implemented*. Abstract specifications are produced using CASE tools with validation support and stored in XML files.

2.2 Code Generation Driven

After the specification is completed, it is used as input for specialized code generators (transformation engines) to produce the persistence, business and user interface layers of the application for a given architecture. For example: MS SQL Server, VB/MTS & VB client code or Oracle 8i, EJB components & JSP web clients, to cite two classical configurations.

On the client layer, we have worked with the following technologies: Visual Basic 6.0, Java/Swing (desktop environments), JSP, ColdFusion, ASP (web environments) & eVB (pervasive computing).

In some of them, we have developed transformation engines (code generators), in others: applications and prototypes.

Depending on the target platform different usability problems have arisen.

2.3 Desktop UI experiences

From our point of view, desktop UIs are the powerful and simplest ones. **Powerful** in the sense that they provide the richest toolkits to build user interfaces. **Simplest** in the sense that are well-known by developers, well-supported by tools and IDEs to create and manipulate such UIs and well documented (there are good books teaching the rationale behind the design of good desktop user interfaces).

Guidelines and style guides are well established in desktop environments. Each platform establishes a style guide (a look & feel) to be followed, making easy some design choices and providing standardisation of UI features accordingly to the style guide.

For this reason, customers easily agree with the general look & feel of the application if it is compliant with the environment and other similar applications.

Nevertheless, sometimes the look & feel of applications generated by code generators does not satisfy the customers.

For example: the form shown in Figure 1 allows to retrieve objects from a database and interact with them: executing services within the objects, navigating to other related objects or limiting the set of displayed objects using filters. The widget used to represent this behaviour is considered like a 'black box' and always has the same appearance (called a *user control* using VB terminology).

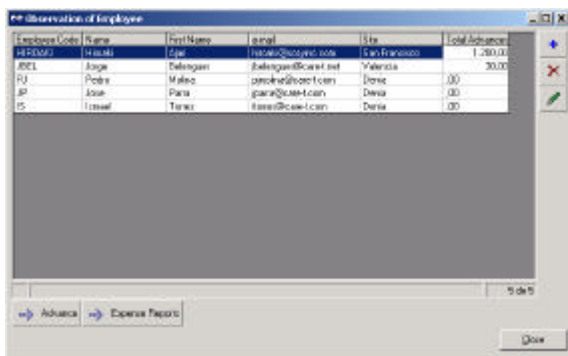


Figure 1. Example of Desktop window.

The customer (or the UI designer) could not agree with this appearance. Maybe, he would prefer to see the vertical toolbar on the left instead of displaying it on the right. Or maybe, he would prefer to view the horizontal toolbar as a list control.

2.4 Web UI experiences

However, Web applications are a bit trickier. There is no single language to develop (you will need at least HTML, CSS, Javascript and server side scripting). Browsers are not compatible with one another, forcing developers to program for and test it in different browsers. UI widgets are of course less rich than Desktop ones. Latency delays and net breakdowns influence your application. Stateless programming is harder than using state and simulating the state using cookies or session management has a non universal solution.

Furthermore, there is no style guide for the web. On the contrary, applications must follow the *look & feel* used in the customer corporate site.

The automated user interfaces are produced with the aim that such interfaces could be the final product and to minimize the manual code. But nowadays, the scope of a web application (number of potential users) is wider than some years before and the profiles of the Internet users are not always well defined. For these reasons, it is more difficult to satisfy the necessities of possible users.

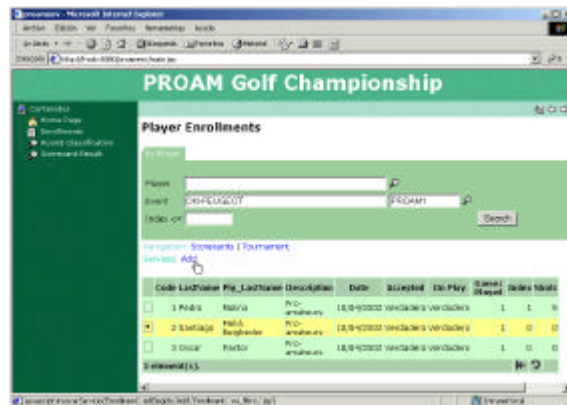


Figure 2. Example for Web environment.

Figure 2 shows the same example as previously shown in Figure 1 but implemented in a web environment.

2.5 PDA UI experiences

The recent emergence of Pocket PC and palm-sized devices compelled us to consider the potential benefits of these mobile, wirelessly connected handheld devices. The main problem with pervasive devices is their constraints in terms of screen size, keyboard, toolkits, memory, etc.

Conventional GUI techniques appear to be ill-suited for some of the kinds of interactive platforms now starting to emerge, with ubiquitous computing devices having tiny and large displays, and recognition-based user interfaces using speech and gestures.

Personal Digital Assistants (PDAs), personal organizers, digital cell phones, Tablet PCs, WebTVs, wall-size displays, large plasma panels and other devices are becoming more and more popular. Interfaces on these very large and very small displays cannot typically use the standard desktop model, and people will not necessarily expect these devices to act like "regular" computers.

The implication of these changes is that we can expect a dramatic increase in the diversity of both the types of computing devices in use, and the task contexts in which they operate. This in turn implies that we are poised for a major change in user interfaces, and with it dramatic new needs for tools to build those interfaces.

Another problem is there is no standard for UI development across different vendors for pervasive devices. Some devices have rich UI widgets meanwhile others have poor ones. Some devices are Internet-capable (have a WAP-Browser, a HDML-browser, a HTML-browser or other kind of browser) meanwhile others are not. Some devices have graphical UIs meanwhile others have textual UIs (or even speech UIs). In fact, some PDA UIs can be seen as a subset of Desktop UI or Web UI (or both), but having constrained screen size and screen resolution.

The code generated for mobile devices applications has the same kind of problems as mentioned in section 2.3. In addition to these problems, we have to consider the reduced screen size of this kind of devices.

	Firstname	Surname	Approved	Consume
	Roberto	Garcia	Yes	04/08/02
act	John	Smith	Yes	05/01/02
act	John	Smith	No	
act	Peter	Keller	Yes	08/21/02
	Ann	Smith	Yes	08/21/02
	John	Stang	No	11/21/02

Figure 3. Example for PocketPC.

For example: the form shown in Figure 3 provides equivalent functionality to that shown before in Figure 1 & Figure 2. Some features are implemented in different forms due to the reduced screen size.

3 The problems found

During these years working on the described context, some problems in this area were found.

3.1 Automatic UIs are good but are not the best ones

Code generation can save a lot of resources and prevent human mistakes. However, full automation can be undesired in the following context:

1. You need to change the look and feel to adapt to a specific platform or application.
2. The computer will not be as creative as a good HCI specialist or UI designer. It will follow a deterministic algorithm to take decisions and place widgets following a fixed layout algorithm.
3. You found a problem that is out of the scope of your code generator. You are neither able to specify nor generate it.

3.2 Recognize different capabilities

Different people have different capabilities. In the same way SE and HCI guys usually think in a very different way. They have different main concerns and solve problems using different approaches. Engineers try to focus on functionality, efficiency, scalability, etc. whereas HCI professionals are more concerned about usability, suitability or funology!

3.3 Different Languages, Similar concepts

We have found that in early stages of the development, the conception of the UI follows a similar approach for each group. Using sketches and wire-frames helps to draw an initial mock-up without being lost in the details. Refining the sketches, tasks, functionality, navigation and contents will appear soon or later.

Is quite interesting to observe how different UI professionals use specialized jargon used by a small group of colleagues.

3.4 Complexity of functionality/design

Small systems are easy to deal with. However, in industrial ones, scalability problems appear on the scene quickly.

HCI and graphic designers tends to treat the functionality of an application as a black box. They do not know how it works and they do not care: a programmer will do, in this case, the hard job.

On the other hand, the same occurs with aesthetics and graphic design: software engineers and programmers will do not touch the graphic design. Otherwise, they could break it down.

For big-sized systems the necessity of splitting functionality and UI design is crucial in order to have

both kind of people working together instead of fighting against each other.

3.5 The not so standard web look & feel

As anticipated previously, there is no standard web look & feel, no consensus of how it should look like. And for marketing reasons, this will probably never be possible: every company on the Internet wants to offer differentiated services respect to the competitors.

Moreover, depending on the user profiles some design options could be good or bad options, for example depending on the expertise level of the users.

This makes harder the web design, in the way that it is necessary to adopt or design a new style guide for the web application to our new customer.

These problems points out the necessity to have code generated easily updateable to support such changes and maintenance. Transformation engines have to be design taking into account the necessity to easily change such look & feel.

4 Some solutions

In our daily work, we have found some useful techniques to deal with these problems. In this section we will describe some of them.

4.1 Separate of Concerns

Separate of Concern is an effective principle to split the responsibilities of different roles during the development. UI Designers and Software Engineers can concentrate in their respective tasks with controlled interference if an appropriate frontier can be clearly drawn.

Patterns like MVC (Goldberg, 1983) or architectures as PAC (Coutaz, 1987) or ARCH (SIGCHI, 1992) helps to implement such a principle for separating UI from functionality.

4.2 Patterns

User Interface Patterns have been revealed as a good resource to be used in UI development: Patterns can be considered as a *lingua franca* (Erickson, 2000) shared by the development team.

Design patterns (Tidwell, 1999) & (van Welie 2000) are quite useful to learn good solution to recurrent design problems.

More over, Conceptual UI Patters (Molina, 2002 & 2003) are also useful in early stages of the development allowing to include users in the requirement elicitation process.

Patterns languages used in a UI also provide homogeneity in such a UI: in the way that the same problem is always solved in the same way and is easily recognisable by the user. Recent workshops addressed this usage of patterns for UI Design (Mullet et al., 2002) & (Fincher et al., 2003).

4.3 Sketches and diagramming

Sketching the UI and making paper diagrams helps to understand the user requirements and to envision the UI.

Papers based techniques (pens, paper and coloured sticky notes) (Ruble, 1997) and (Constantine & Lookwood, 1999) are useful during the requirement elicitation phase to work directly with customers and stakeholders.

Wireframes techniques (van Welie, 2001) or UI computer supported sketching like in DENIM (Landay, 2001) have the advantage that eliminates all distracting design choices and concentrates in the negotiation of an abstract UIs to focus the discussion in what it is needed instead of how it could be implemented.

All these documentation artifacts are quite valuable for engineers and HCI professionals. They can work using these materials to identify and resolve pending ambiguities and problems.

4.4 Mock-ups and early prototyping

Prototypes and mock-ups are also quite helpful. Users can test the system in early stages before it is built. Users, HCI professionals, and engineers can take advantage of mock-ups to detect usability issues even after the final product is built.

4.5 Functionality driven development

What we call a "*functionality driven development*" usually starts trying to find all the functionally need and later on, organize such features in a suitable user interface.

In this approach, the functionality is ready before the UI. It is useful when designing well-known applications that should follow a strict style guide. Transformation engines can be used to produce a high percentage of the final code. If 80% of the generated UI scenarios are ready to use, the saved time obtained by the usage of code generation can be used to redesign or tweaking the UI of the other 20% pending scenarios.

The need to change the look and feel of the UI will hardly ever occur.

4.6 UI driven development

On the other hand, a more “*UI driven development*” starts identifying user tasks, scenarios and prototyping the user interface. In this process, the functionality is incrementally discovered by HCI specialist, interaction designers, etc.

The UI is built manually and it is ready before the functionality is implemented. Code generation is less suitable for these tailored UIs, however is still very valuable for the functionality of the application.

4.7 Templates

Templates can be an effective way to implement SoC (Separate of Concerns). Static templates (applied in design-time) like the used in Macromedia Dreamweaver (Macromedia, 2002) or dynamic templates (applied in run-time) as introduced in Velocity (Velocity, 2002) or Smarty (Smarty, 2002) helps to maintain contents, behaviour and presentation in separate files. Using templates, in this way, can be an effective way to maintain separately the design and layout of an UI from the functionality behind it.

4.8 Wizards

In “*UI driven development*” the UI designers do not need to know too much about the implementation of the functionality. They only need to invoke such behaviour in certain points during the user interaction.

In this scenario, Wizards tools driven by functional specification metadata can help to select the functionality needed for a scenario and provide the code in a particular language to invoke such functionality. Such a code can be inserted automatically in the UI design after the designer choice.

Doing it in this way, designers will only need to know what functionality they need and how the invokers work. Meanwhile, the implementation will be kept as “top secret” by SE specialists.

4.9 Customisation

Static applications are more rigid than configurable ones.

Despite the user profiles and differentiated user interfaces, each user can be seen as unique.

In this way, if we can also provide generic customisation mechanisms in an automatic way, we will allow users to personalize their application to their needs.

5 Conclusions

The problem stated by this workshop is not an easy pie, on the contrary is a real problem nowadays we must cope with. The possible solutions proposed in this position paper are probably not enough to solve the general problem in the area. However, in our domain our two cents seems to help in our daily work.

We hope these ideas promote discussions during the workshop and we are looking forward to getting feedback about that from the rest of the workshop attendees.

6 References

- Constantine L. & Lockwood L. “*Software for Use. A practical Guide to the Models and Methods of Usage-Centered Design*”. Addison-Wesley, Reading, MA, USA, 1999.
- Coutaz J. “*PAC – An Object Oriented Method for Dialogue Design*”. In Proceedings of the INTERACT’87 Conference, Elsevier Science, Stuttgart, Germany, 1987.
- Erikson T. & Thomas J. “*CHI 1997 Workshop. Putting It All Together: Towards a Pattern Language for Interaction Design*”, 1997. Summary available at http://www.pliant.org/personal/Tom_Erickson/Pattems.WrkShpRep.html.
- Fincher S., Finlay I, Greene S. Jones L., Matchen P., Molina P.J., Thomas J. CHI-2003 workshop: “*Perspectives on HCI patterns: concepts and tools*”. Fort Lauderdale. USA. 2003. <http://nitro.watson.ibm.com/chi2003Workshop>
- Goldberg A., Robson D. “*Smalltalk-80: The Language and Its Implementation*”. Addison-Wesley, Reading, MA, USA 1983.
- James A. Landay and Brad A. Myers, “*Sketching Interfaces: Toward More Human Interface Design*.” *IEEE Computer*, vol. 34, no. 3, March 2001, pp. 56-64.
- Macromedia. *Macromedia Dreamweaver MX*. <http://macromedia.com>, 2002.
- Molina P.J., Meliá S., & Pastor O. “*User Interface Conceptual Patterns*”. In Forbrig, Limbourg, Urban & Vanderdonck (editors), “*Proceedings of the 4th*

- International Workshop on Design Specification & Verification of Information Systems DSV-IS'2002", Springer Verlag, Berlin, Germany, 2002. ISBN 3 540-00266-9. In Lecture Notes in Computer Sciences.
http://www.springer.de/cgi/svcat/search_book.pl?isbn=3-540-00266-9.
- Molina P.J. "User Interface Specificacion: from requirements to automatic code generation" (In Spanish) PhD Thesis. Technical University of Valencia, Valencia, Spain, 2003. Available at: <http://www.dsic.upv.es/~pjmolina/EN/thesis.html>
- Mullet K., McInerney P., van Welie M. CHI 2002 Workshop. "Patterns in Practice: A Workshop for UI Designers", Minneapolis, USA, 2002.
- Ruble D.A. "Practical Analysis and Design for Client/Server and GUI Systems". Yourdon Press Computing Series, 1997.
- SIGCHI. The UIMS Workshop Tool developers: "A Metamodel for the Runtime Architecture of An Interactive System." SIGCHI Bulletin, 1(24), pages. 32-37, 1992.
- Smarty, PHP template engine. <http://smarty.php.net/> 2002.
- Tidwell J. "Common Ground: A Pattern Language for Human-Computer Interface Design", 1999. http://www.mit.edu/~jtidwell/common_ground.html
- van Welie, M. "Wireframes" 2001, <http://www.welie.com/articles/wireframes.pdf>.
- van Welie, M., van der Veer G., Eliëns A. "Patterns as Tools for UI Design". In International Workshop on Tools for Working with Guidelines, pages 313-324. Biarritz, France, 2000.
- Velocity, Java Template Engines. <http://jakarta.apache.org/velocity/> 2002.