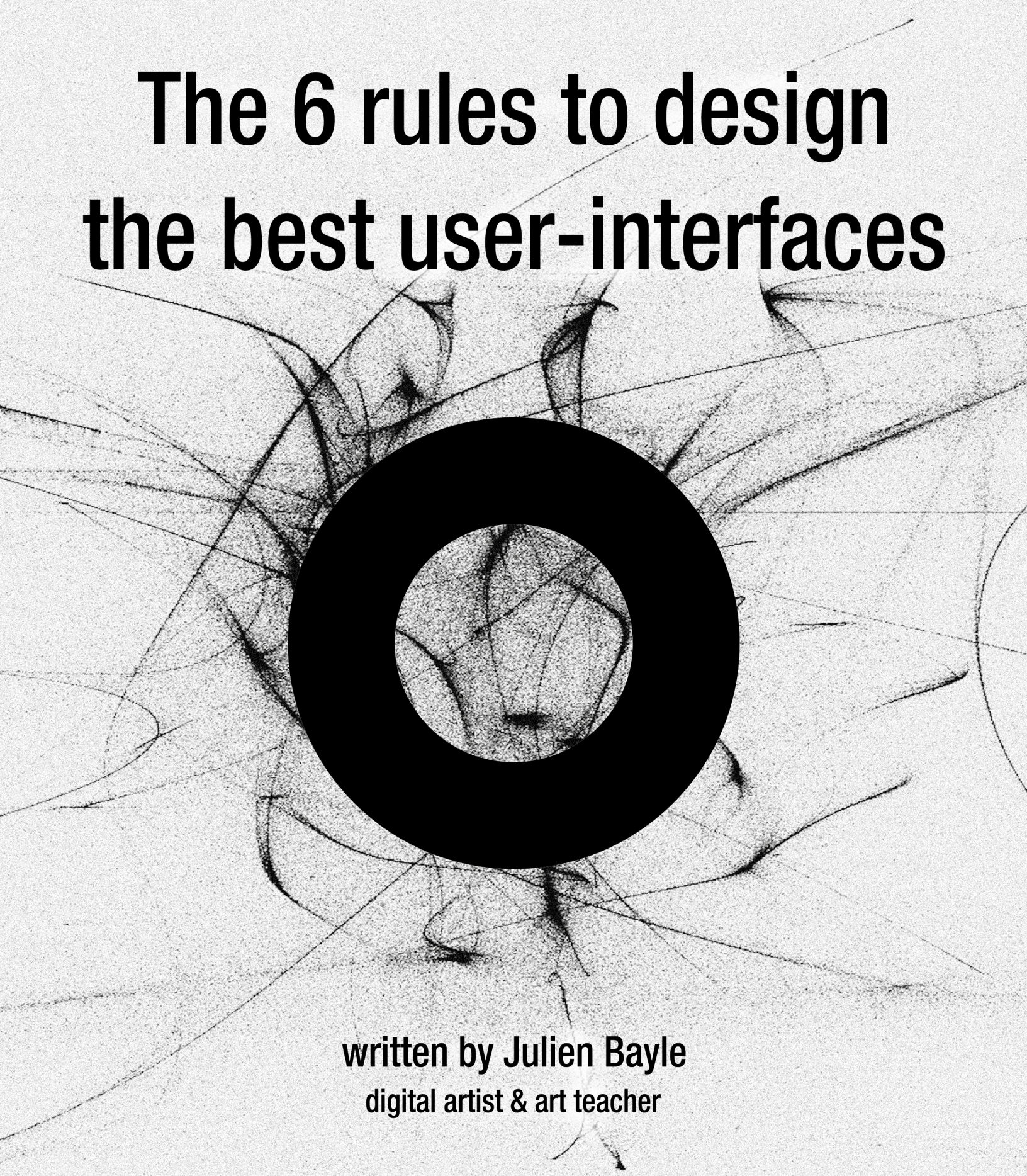
The 6 rules to design the best user-interfaces



written by Julien Bayle
digital artist & art teacher

The 6 rules to design the best user interfaces

Julien Bayle

This book is for sale at http://leanpub.com/the-6rules-to-design-the-best-user-interfaces

This version was published on 2013-12-24

Tweet This Book!

Please help Julien Bayle by spreading the word about this book on Twitter!

The suggested hashtag for this book is #julienbayle.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#julienbayle

Also By Julien Bayle

Max for Live Ultimate Zen Guide

Le Guide Ultime et Zen de Max for Live

Contents

About

You can find my bio here[1] I'm a french digital artist and art teacher. I'm also an Ableton Certified Trainer and Max6 programmer.

I wrote Max for Live Ultimate Zen Guide[2] and published it in 2013.

But, I wrote this one in 2010. Schematics have been designed from my drafts by Manuela Ciancilla and the book itself have been proofread by Glenn Reuther. This short guide had been originally published in 2010.

I especially let it exactly as it was for 2 reasons:

- it was my first ebook and I wanted to keep the first version of it
- I wanted to see how much I could have changed since this very first ebook

[1] http://julienbayle.net/bio/

[2] https://leanpub.com/Max-for-Live-Ultimate-Zen-Guide

Introduction

User Interface Design is the art of building the elusive missing link between you and the system you want to control and then hear the desired results.

One of the biggest challenges in user interface design is to keep the user perspective as the focal point of any **Interface Design project**.

As a case in point, just take a look at the grand majority of popular user interfaces, and how they are commonly used.

Once you do, you'll begin to understand that a lot of companies (even the biggest ones) seem to have forgotten this main idea.

They inevitably cut corners to keep costs low, while sacrificing this crucial element; they fail to see the forest for trees. Many owners of these devices have to create work-arounds in their processes just to accomplish half of what they hoped the device would do, after they shelled out their hard-earned cash for it. This goes for software even more. Even then, they often fall woefully short of expectations.

I find it hard to believe that this process is so often overlooked when engineering these device designs. It's like forgetting to include the design for a toilet in a spaceship; kind of important. Don't you think ? Kind of embarrassing too. :-P

I have always been interested in interfaces; from cell membranes in biology, to writing software applications and hardware drivers.

Ultimately, in the performing arts, which requires specialized tool to control complex creative systems, each piece on which I work is an interface to control some system.

For me, the best user interface is one whose design provides a rich and complete tool which enables the user to easily accomplish their tasks; one in which the interface itself becomes invisible, when its user is able to concentrate solely on the process.

Otherwise, it becomes a distraction. In this way, the user is directly connected to the system, because the interface has become an extension of the performer. Performer and system are then one.

Speaking on the level of the most basic equation, on one side is a human being. On the other side is any other thing in his/her world, but in between each is the interface (as well as a medium through which it is understood and used) which provides the connection. It's a critical component, but the one that is most taken for granted. What would it be like if any of the following were missing from your equation:

a common language, network, postal services, signage, cars, and even more elementally, air, water, gravity… Also, an interface can serve as a translator between two things which don't communicate in the same way.

From your mobile phone to your car, from the pull-down menus in a software application or even the application itself, or from the DJ console to this matrix of buttons or pads so many make music with, everything is interfaced. Humans travel in a world filled with interfaces to all systems he/she has to interact with. Considering this point, imagine all user interfaces were correctly made, were all useful and even efficient.

Again, if we look closely enough at those already existing, we see this isn't the case. Even big companies with large R&D budgets, and a fully qualified staff of engineers, who may have all studied user interface design at school; bad user interface design is still widespread.

What does bad mean ?

Bad equals any or all of the following :

- ugly
- inefficient
- unintuitive
- too general
- too precise
- inconsistent
- too small
- or even too cryptic

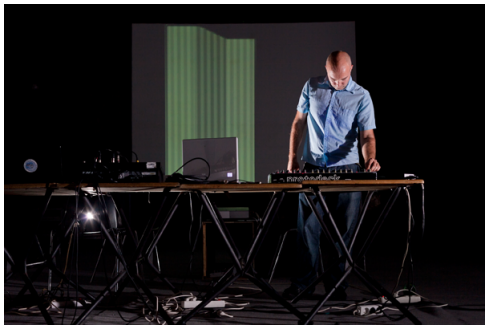
Understand that building a good and usable interface is a complex task, requiring considerable thought. Knowing this you have to come to the realization that even if it is complex, **it can be done**. If large corporations can't do it, **you can do it**.

And you're lucky: I offer you the opportunity to break the vicious cycle driving the designs of those bad interfaces. You can really make the difference by building your best user interface ever !

In the next few pages, you'll discover the principles and methods which I follow when designing and building large and complex hardware interfaces such as **Protodeck**[3] ; you'll come closer to visualizing the hardware controller of your dreams, and all its interfaces with the final application you wish to control and get feedback from, as I have.

I designed and built the Protodeck controller, as well as the BONOME controller (the more widely known of the two), for use both on stage, and in the studio.

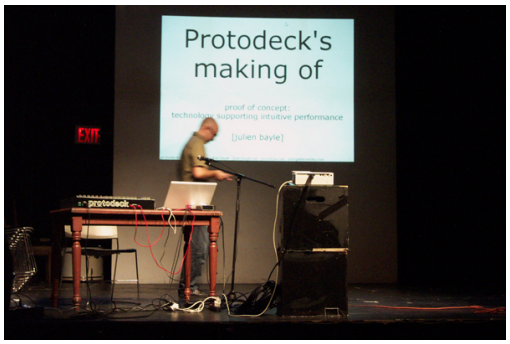
[3] http://julienbayle.net/works/creation/protodeck-midi-controller-for-ableton-live/



Protodeck controller on stage

They are both hardware interfaces designed to have remote-control of the Digital Audio Workstation called Ableton Live. As I was going through the various stages of their development of these devices, I documented the journey on my blog[4] .

I have since been invited to do presentations in several European cities, and most recently New York City, to showcase my story, the how's and why's I felt it important to realize these designs, and to share my knowledge and experiences.



Protodeck's lecture in New York, US

I often use examples in the field of music controllers because it is very easy to illustrate.

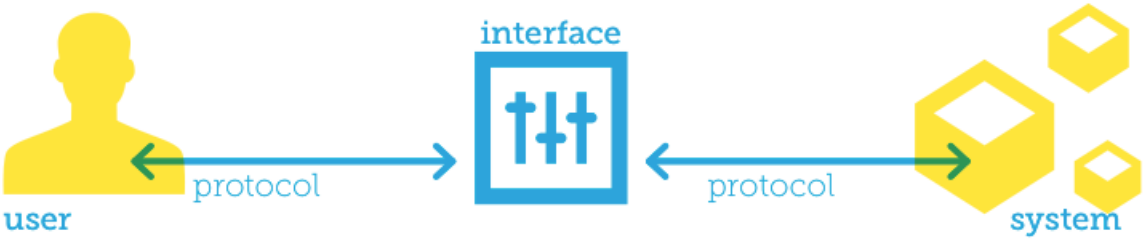
Before we go any further, remember :

- **don't focus now on details**
- **take a breath**
- **free your mind**
- **go to the next page!**

[4] http://julienbayle.net/blog/

1 Some pre-requisites

A user interface involves a User, a System, the Interface itself and some protocols arranged a bit like this



- The **User** is you (or folks you target for your interface design).
- The **System** is the tool you need to use in the most ergonomic manner possible.
- A **protocol** is a standardized set of rules previously decided for making different entities communicating.

You want to be able to control the system and get some sort of feedback from it. The interface facilitates usage of the system, because it is the translator between the system and you.

This concept is very important. An interface can translate many things into many other things. Basically, you have to understand that an interface sends messages but must convert them from one **protocol** into another protocol so that it's understood at the other end. As an example, the interface can convert (one of the possible translations) a linear movement of a finger into a logarithmic numeric variation of a parameter. (Translation : use a volume slider). It could also translate it into an analog voltage which has a value.

As another example, an interface can understand and convert messages from/to a world talking MIDI protocol into/from a world talking OSC protocol (OSC and MIDI both being protocols used in the worlds of music & multimedia) to make music machines communicate amongst themselves.)

We'll also talk about other concepts, but we'll describe them step-by-step and not in too much in detail, in order to keep it simple.

Now we can move further and dive inside my protocol for developing these interface devices.

2 Listen to the system

Don't by-pass control & feedbacks. Ever! Or how to avoid the big by-passed feedback mistake? A very common error to avoid in user interface design is to leave out the direct link between the CONTROLLER and the SYSTEM.

What is the problem ?

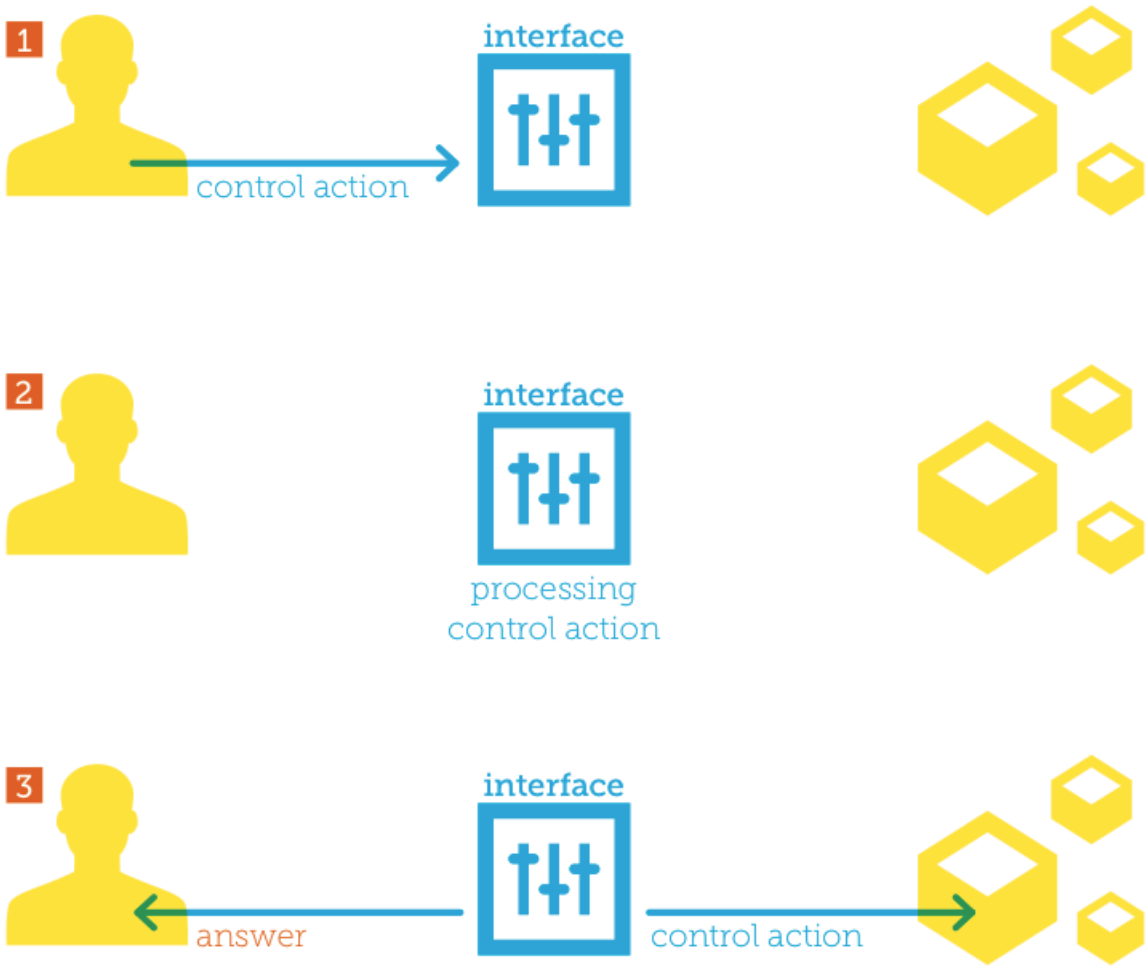
The answer comes from the interface and basically it has to come from the system.

As an example, the user initiates the control step (step 1) by pressing a button on the interface.

The interface then processes that command and follows its instruction to send a control action to the system.

Here's where the problem comes in though.

Instead of the system issuing the answer (feedback), many interface designers have the interface issue the feedback step (step 3) just because the user pressed the button, even before knowing if the system has safely received the control action, or is even connected to the interface !



The wrong feedback mistake

Here's how it's supposed to work

1 The **control** step is the user pressing the button, initiating the request.

2 The **action** step is the interface processing and passing on the control action.

3 The **feedback** step is illuminating the led that indicates that the instruction which the button press initiated, has been received by the system.

Control (step 1) sends the **action** trigger (step 2), which waits for **feedback** (step 3) from the system to determine whether the indicator should update status.

Interface **trigger** (control) sends to » **system** (action) sends receipt to » **Interface indicator** (feedback).

Many designers make the design decision to press a button and have it illuminate the indicator before the system receives the request and performs some action. But, lighting the indicator should

be the indication that the **action** (step 2) request has entered the system, and not just that the button was pressed. They choose to put the cart before the horse: they often design it as the feedback to the button press, and not as a receipt that the system successfully received the request. A wrong interface design is one that provides the same feedback, whether it's communicating with the system or not, and if so, then it is providing false feedback.

I guess this might happen because designers want to test the hardware interface mechanisms so they like to check to make sure the light works more than to know the success (or failure) of the communication and action. That's one way of carrying through with the manufacturing process, but not for quality control. They then should remember to alter the code to make sure the confirmation is coming from the system, and not the interface. They just don't though. I guess they have a lot of confidence in their code, and don't mind misleading the user.

It was just a bit complex in the case illustrated above, but it made for a functional demonstration of a poor interface design, rather than a useful box.

The feedback concept is important. **Feedback** is a word that can be replaced by answer after an action. Doing so guarantees that communication has been established, and that both sides are listening and responding. Remember those folks who only speak without listening.

In the end, they are just listening to themselves. It isn't a dialog; you cannot interact. In order to create a dialog, they have to listen.

Here, you have to listen to the speaker, I mean, the system. The interface is only the translator, not the speaker ! Don't shoot the messenger!

One day, a guy contacted me in chat and we talked about his interface design. Because the feedback processing was a bit tricky (protocol conversion was awful in this case), he by-passed this part and wanted to make an led blinking immediately after he pushed a button.

I told him: "please don't do that!" ; his answer was: "no problem, I'm sure my action will always be ok, and I want to mute this track like that and have the correct led light up." When he tested that and understood that his liveset (in Ableton Live) was totally separate from his led matrix hardware, he just spent a bit more time on the feedback process. Because without this part, the whole status feedback part of the interface was … useless, and you don't want to build something which is unusable, right ?

Well, at least that makes two of us: you and me! :-)

3 The user is the core

Put the user at the core of your project! or how not to let technical constraints drive your mind ?

Sometimes, I am called upon to review new (commercially released) interface designs here & there. I'm viewed by some companies as "the guy who is never happy" with their interfaces, again, because they so often seem to miss the point.

Indeed, as a frequent beta tester, I have some systematic reactions when I'm testing a new interface. When I'm doing that, I'm just a user; certainly a hard to please user, but a user all the same.

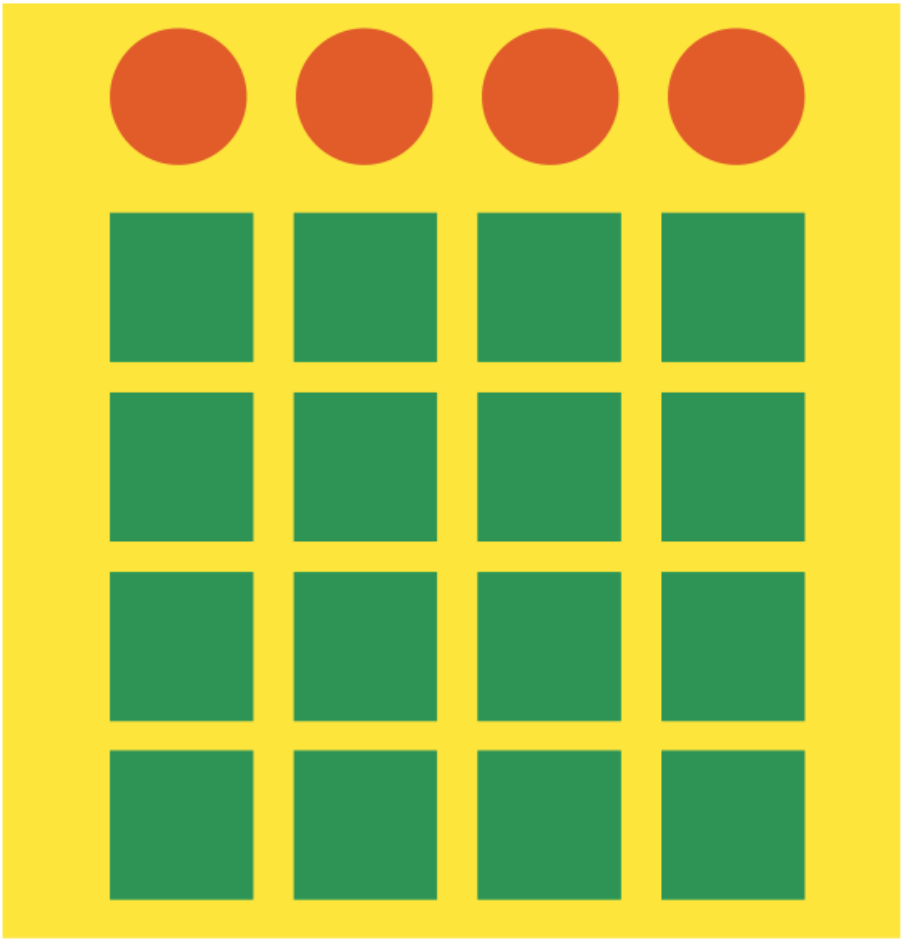
I would like to show you some interfaces I discovered (again) in the music controller world. Those examples are commercially available controllers!

Monotonous Monochromatic LEDs

This is a phenomenon I discovered, which is difficult to illustrate, but very famous and very weird problem. This refers to all the indicator lights being the same colors. Just put me to sleep while I work with your interface. :-P

On behalf of all of us users, I asked "WHY?" and, you know, I had more than enough answers to my question, but very annoying and inexcusable answers, at least from the end-user perspective. I guess they're perfect answers if you design but never have to use. I'm sure you'll recognize them.

Incomprehensible Mis-Alignment



The “Incomprehensible Mis-Alignment” problem

A perfect column of buttons, with a single knob at the top, and “just a little bit” off-center.

About The “incomprehensible mis-alignment“ problem, I had:

– “ Oh man, you can easily use the first knob for the first column of buttons” ; no comment

– “Those knobs aren’t only for using them linked with the column” ; indeed. But we, users, are using them like that, so…

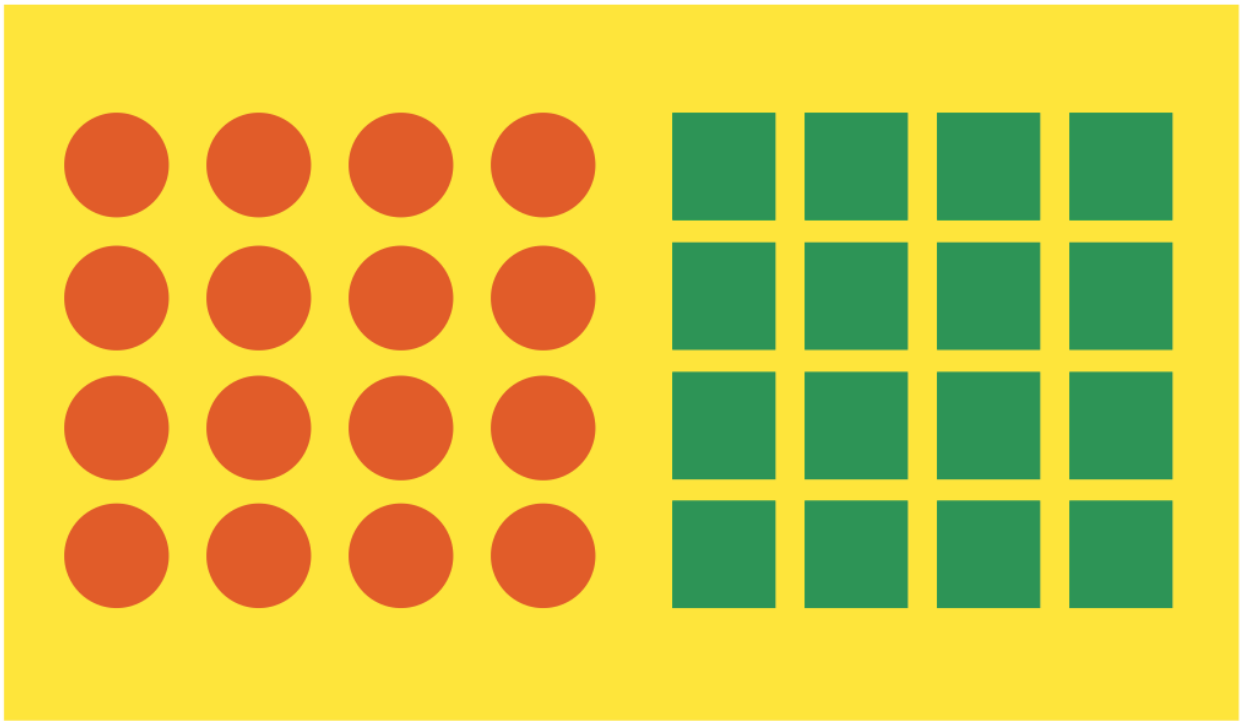
The real problem was about printed circuit boards which had been badly designed, considering the space in the box!

It was a purely technical manufacturing problem, a design mistake that drove to making a misaligned interface design.

Another result, less important, is the aesthetic: it looks ugly like that. But it isn’t the most important,

I agree, compared to the ergonomy! Ergonomy has to be considered firstly, because it facilitates the user work.

Where Is That Track ?



The “where is that track” problem

About The “where is that track” problem, I had:

– “oh man, noone uses this interface like you !!” ; they said I wasn’t a common user.

– “if we had put knobs above, the interface would be too big” ; at least, it is honest to talk about their technical constraint :)

Every user I spoke with told me they use the interface… like me (which is very nice for me to figure out, because I understood I wasn’t so weird after all); each column represents a track (often in Ableton Live… again this one !). So… you have to count from left to right… 1…2…3, ok this is the track 3, then . . . (or is it from the bottom up ?) It fails! The first thing I would do if I had bought this interface: use it after a 90° rotation Clockwise. Ergonomy has to consider users, at least those under the bell gaussian curve.

About The “monotonous monochromatic leds” problem, I had nothing else than: – driving rgb leds is more difficult and it involves more power, more code in the firmware

This time, I’ll give you my answer to their one: work smarter and find the solution!

Indeed, users need to have more than 2 states leds. Ok, we can have 3 states with monochromatic leds: on/off blinking. But we, users, need more on stage. Because we want to focus on the result: the art we're trying to create & perform. 2+blink states aren't enough. End of story.

Ergonomy has to give the greatest flexibility to users. So, considering the two previous problems, even if they have technical constraint, they should think about that: they have these big constraints because they work for humans in order to facilitate their life and in order to remove their constraints.

This is their job : to relieve constraints and resolve to make the users' job to use products without constraint! It involves a change in thinking : working a bit smarter, designing for simpler user process.

. . . and less living in a state of denia !

With some companies, I also discussed working on interface design like a multimedia controller. Here's a tip, (and this could be the main tip) : be the user ! Use it, or lose it !

Project managers often forget, at each step of their project, to sit in the user's chair. It is usual because they are saturated by other tasks, considering cost challenge, time & schedule, pure technical considerations etc.

But the best one sits each day in the user's chair. If you sit in the user's chair, you'll design better interfaces !

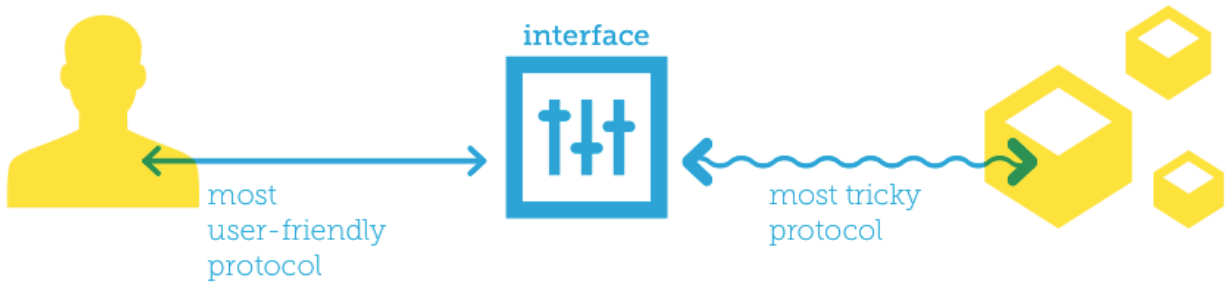
4 Hack protocols

Or how not to waste time coding a protocol from scratch ?

Remember? We need protocols between you & the interface, and between the interface and the system. The first link (between you and the interface) seems to be the clearest to understand because of the previous chapter.

Indeed, you have to build an interface for your fingers if you have fingers, with blinking leds if you have eyes, etc. So the protocol between you and the interface SHOULD BE as intuitive as possible ! It is the trivial side of this chapter.

The big part is « the other protocol »: the one between the interface and the system, and I could briefly describe that by the following schematic.



It also means you'll spend more time to design/code/build the Interface-to-System protocol than the User- to-Interface protocol.

A lot of people, especially those learning how to make this kind of gear, wants to build their own protocol. (OH MY GOSH !)

It is very nice to do that. (tick tick tick tick . . .) But I want to give you another big rule, based on my own experience with the *protodeck* controller: There already exist A LOT of protocols you can use, and hack! To hack doesn't mean here the same thing you hear on the radio about computer hackers who break the code to steal your paypal money!

The method is almost the same, but the purpose is far from it ; I can define it like this : a hacker modifies things to make them fit their needs better. It's custom design.

"Necessity is the mother of invention"

It would make you another Do-It-Yourself activist, but if you're reading this, it means it's too late : you're already are… so another step in that direction will only make you stronger than ever!

Opening up a device and taking a peek at what's inside makes you stronger than IMAGINING what's inside and trying to build a different boxfrom scratch, right?!

Using an existing protocol means:

- standard exists and are already published and stabilized
- programming library probably exists for your interface and system
- schematic probably exist for your hardware (if your interface project requires that)

You'll be 100% winner!

For the Protodeck controller, I used the MIDI protocol when a lot of people advised me to use OSC. OSC is great. It goes further than MIDI. But when I built the protodeck, I couldn't get OSC to communicate easily (meaning, reliably) with my system.

I needed to send/receive 2 types of information:

- 0 & 1 (kind of binary information)
- from 0 to 127 (kind of analog information)

An already described and documented MIDI protocol was interesting because it provides exactly that. I encourage you just to read that. Basically, the protocol describes the hardware and the protocol under the hood. MIDI note message are coded with 3 parts (called bytes by those g33ks we are):

- the channel
- the pitch (the tune of the note)
- the velocity

MIDI control change :

- the channel
- the control change number
- the value

On the protodeck controller:

- when I push/release a button, the protodeck sends a special MIDI message called "a note" (push = note-on, release = note-off)

- when I turn a knob, the protodeck sends a special MIDI message called “a control change” (with a value from 0 to 127)
- when I send a MIDI note on to the protodeck with particular velocity, leds on the protodeck get brighter with this or that color, or are switched off.

Using this sort of coding which are very natural (I mean, it isn’t as supernatural as it seems), I had only to use MIDI libraries/modules on each side:

- on the protodeck,
- on the system

Using libraries means: I already have the whole program to send and receive (process) MIDI messages. I only had to write sendMidiNote instead of a complex cryptic code ! While you weren’t looking, we’ve just entered a bit more in the flesh of our passion! :-)

5 Be minimal

Or how an old genius guy named Leonardo Da Vinci, can still help today ?

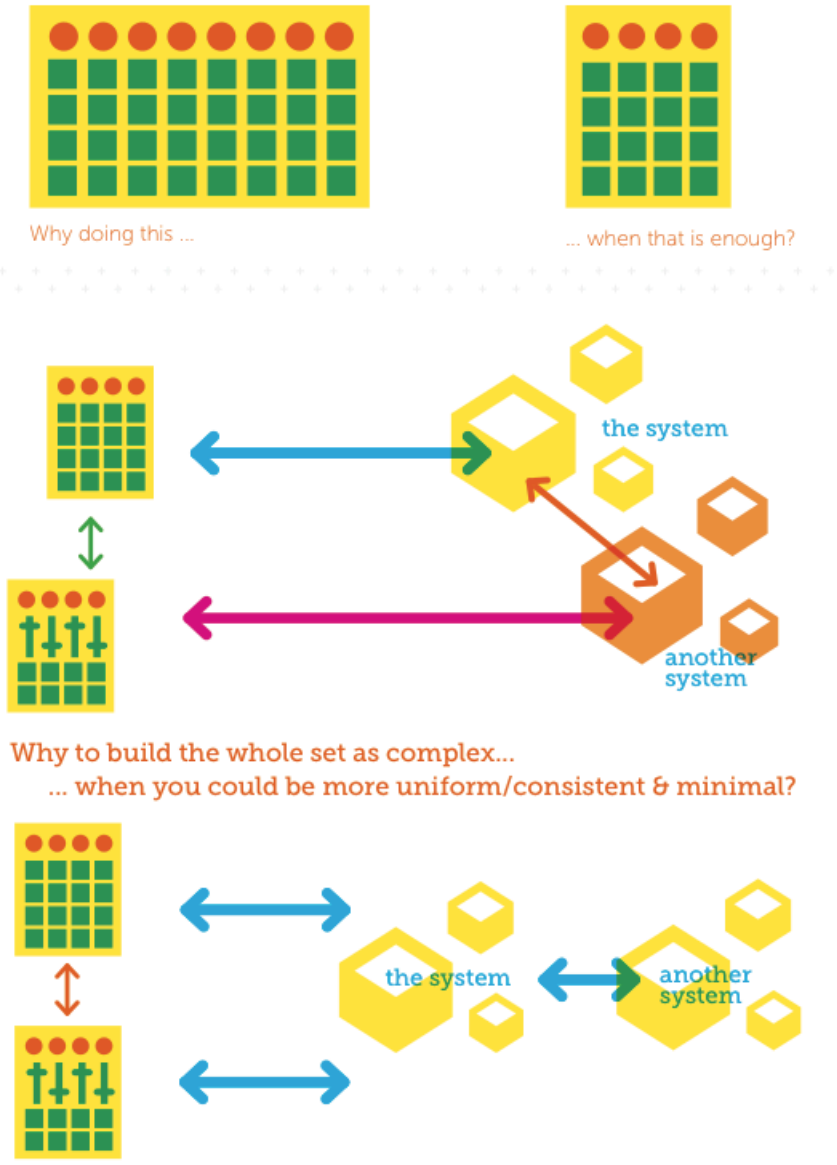
Even if minimal stuff seems more simple at the beginning, it often involves a long path that has to be walked progressively. Indeed, in order to know exactly what you need, you need to progressively reduce the big initial setup down to the very singular core of what you need. But targeting a minimal design at the very beginning wins!

In creating handled & reduced constraints, it increases liberty. If you need to create music and perform it live without being distracted by a too big choice in your Ableton liveset, for instance, then you have to reduce your track numbers.

I did it and experimented with that in my life, because I'm a real "Principle of parsimony" addict. (Ockham's Razor). I never produced as much as I'm doing now before I decided to reduce my track to *only 8. It could be dramatic for people to reduce things. It is dramatic for me, too! Really! Often, they feel constrained (but most of them never uses more than 6 tracks even if they can use 130). Others are "afraid" to be short; those individuals need more than 15 return tracks, too.

At last, a lot of my students need more because they don't know the tool enough. So they make workarounds everytime. After hours of practice, they often reduce too! Here, I offer you my experience of studio, stage and of people I have met.

Reducing & being minimal around the core of your real need definitively wins.



This principle can be applied at any scale. From this precise function of your interface to the whole interface itself and from the whole system of interfaces to … your life.

In the example on the left-top of the page, I wanted to express the fact we're all focused on.

"Would I have enough elements if I choose to evolve in the future?"

The problem is: "Planning is guessing" as 37signals' guys wrote.

You cannot know what your needs will be in the future. So, why build an interface with 856 buttons if today you only need 100 ? It would be better to build one with around 100 buttons. 108 maximum.

Believe my experiences in studio & on stage, if you target n number of buttons today, if you have n+8, you'll only use … n. It's like that everytime.

In this example, I want you to understand the minimalism from another higher point of view: at the whole setup.

3 tips:

- use the same protocols as often as possible
- avoid complex routing as often as possible
- use hubs to aggregate/split flows as often as possible (here, this is The System)

And be careful & don't answer: "it doesn't work in the real life"

Because firstly, it does. And secondly, it would be an excuse not to go further and simplify the system. Don't forget, each hour spent working on it now saves in freedom of creativity for the rest of your life!

6 Stupid box & smart computer

Aim for a stupid box, but a good computer. Or how to get benefits of technology building unsmart interface ?

Today, CPUs and memory are cheaper than ever. We ALL have powerful laptops or desktops (or cell phones, for that matter !)

A lot of designers began their project with a pure interface in mind.

And, progressively, they includes other ideas like:

- a sequencer
- many protocols
- a server inside And much more.

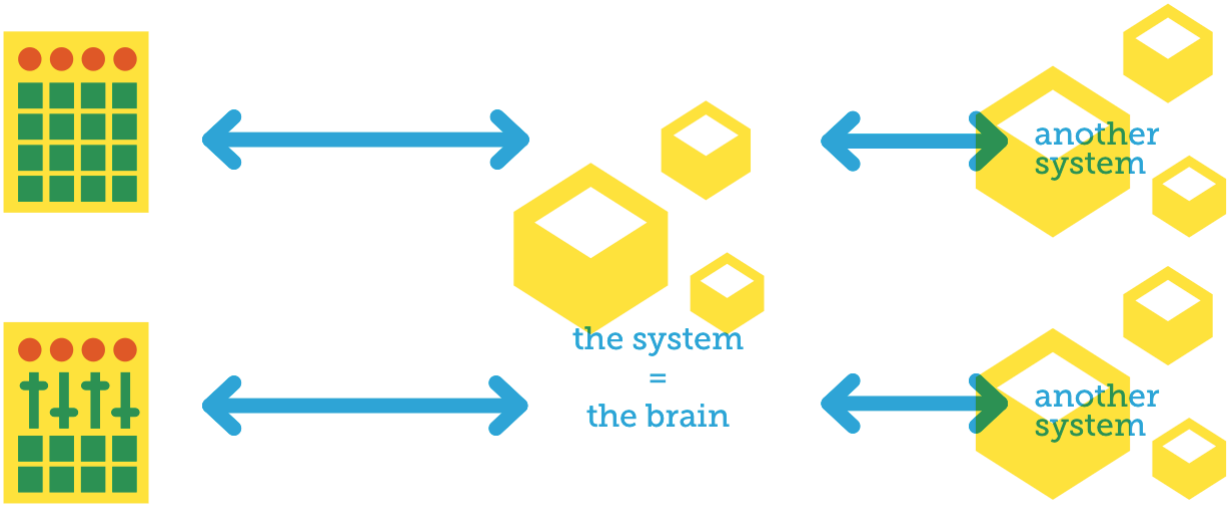
Firstly, they probably didn't read the chapter about minimalism in this eBook.

Secondly, they made the big mistake to mix interface & system/machine.

Our interface can interface with everything today. Users need to quickly change, merge, update their configuration and making that on the computer side is much easier.

So, here is my advice: Build & Target stupid interface, because the computer is the brains of the operations !

I'm sure some of you starting to hate me, thinking, "Man, I want an autonomous interface" 99% of you need an interface to use with a computer, precisely!



And what should you put inside the box ?

The minimal firmware to handle 3 things:

- your inputs (buttons, potentiometers, encoders, pressure sensors, and any other CONTROLs (in the same way I describe since the beginning)
- output (LEDS, motors for faders, and any other FEEDBACK)
- protocols, so there's no misinterpretation of what the user wants.

No more, no less. Only the basic functions required to use the hardware.

You need a sequencer ? Put it on the computer.

You need a monome simulator ? Put it on the computer.

When you push a button on the interface.

The interface should NOT send a message like "activate this precise function inside the targeted software tool." It should say " button47 » or something simple like that.

But I hear you say you need the OSC protocol ? Put it in the firmware.

You need to handle this nice ribbon controller ? Put that in the firmware too.

All hardware handlers MUST BE in the firmware. For those specialists who may read this: don't unbounce your buttons inputs outside of the firmware! It is just another example among many others!

And I'm going further: if you need to use your interface with other devices (I mean synthesizer, drum machines, anything other than a computer), you should really interface them through your computer!

I can redraw here almost the flowchart as before in the previous chapter with interfaces at the left, The System in the center.

The System is the brain, the computer.

Do this and you rule all of your machines in one aggregating point.

The system :

- is controlled by the interface
- is used as a big router which can address messages from/to anywhere
- is easily backed up (you'll have ALL your configuration in the same point)

I can use a nice, concrete example: max for live & ableton live message routing.

You have interfaces, external musical machines (synth & more), and your system.

If you build a software interface on the system which can:

- handle your hardware interfaces
- handle the other systems.

You have a very powerful meta-system and you'd take all the benefit of your computer in that case.

I made an interface like that with max for live. It handles the protodeck, but it also handles my other interface, including the monome.

7 Cutiness is nice, you need more

Or how not dive blindly in this multitouch's world? Today, new tangible interfaces are available in every part of your house.

From your phone to your tablet, you can directly drag/ grab displayed objects. Of course, this technology has been hacked, tested, used very early by those impassioned g33ks in order to make them fit well with their needs (customized/personalized solutions).

And the big error came with the 2D gestures, very often poorly implemented because they are used to replace everything else. We used to see new technology replacing the old ; here old would be 3 to 4 years only.

All these interfaces, even haptic are nice, handsome too, but not enough yet, and probably not enough in the future.

Why? We're in a 3D world! We need 3D interface and we need to « reach out and touch something. » (This is a slogan I quote from AT&T used for humor.)

If I want to push a button, I have to make a vertical movement. Ok, it can be made. If I want to push a fader, it can be made too, even with no haptic feelings.

But we ALL need knobs. I'm not sure this need will end in the next few years, and knobs are just simulated with those multitouch systems.

The 2D interface gives a limit here if it is used alone. I'm not the only one with this problem.

A lot of designers working in the audio\studio field work on it, and have just began to include real knobs on screen (ie., the screen surface is drilled and fixed knobs are included in order to display knobs information around them).

This is the proof that users need more.

So you have to think about that and even if you want to include a multitouch screen in your design, don't forget all of that! . . . and please, (and this could be the last word before the big conclusion:) you'll need more than an unreliable WIFI connection!

This message is put here, and not in some other place because it talks about the tablet.

So please, hack it and use OSC messages, but do so inside a wire, with a cable connected. You'll thank me for sure after some gigs on stage in various conditions.

8 Just for remind you

I wanted to remind you of the different topics of this document.

1. **DON'T TRUST THE BUTTON** means the interface is just the translator
2. **THINK USER** means the core of your interface project is the user for whom you make it
3. **THE STUPID BOX AND THE SMART COMPUTER** means your computer have to be smarter than your box
4. **HACK PROTOCOLS** means you would do best to avoid reinventing the wheel
5. **BE MINIMAL** means you would do better to avoid the race to the most
6. **CUTENESS IS NICE, BUT YOU NEED MORE** means you'd better be wider than iPAD.

Throughout this pdf, I hope you have realized and figured out a lot of concepts and new ideas.

I picked a fight a bit with some interface builders I could quote here. But, I already had discussions with them, so they know what I think about their stuff.

I wanted to offer you my opinion, today, in the early of 2011 about interface design. I wanted to tell you: "YES, YOU CAN"

Don't believe others who say, "Wow! It's too difficult" or "a lot of people already made that".

YOU are your best power, yourself !

YOU can make and build things without waiting for others to build something less for you.

The future is YOURS. Don't let them make it for you !!

So, make it, and make it now !