

Why We Should Not Be Here and What We Should Be Doing Instead

Kenny Tilton
kentilton@gmail.com
circa ECLM 2008

Information matters, so how well we process information matters. We process information with software, so how effectively we program computers matters. Not one person in this room has any doubt that Lisp is far and away superior for building software, nor doubts that everyone should be using Lisp for everything.

SAIC does not use Lisp. SAIC is the gigantic military-industrial entity that failed in two tries to automate the FBI. Had they not failed 9/11 does not happen, because it almost did not happen anyway: the FBI actually had enough information about strange people from the Middle East taking strange lessons on flying airlines to have stopped it, they just did not put it together because the information never got connected.

If the FBI has better systems, thousands do not die. If Florida has better systems, the wrong person does not become President and invite Mr. Cheney to kick off a war to make Haliburton rich. No, the election should not have been that close, but with the American people voting as mindlessly as they are for the love of God if an idiot like Bush loses an election let's make sure he does not win it. Systems matter.

If the FAA could build software we would not have the current greatest threat to airline safety, namely runway near-misses. They can't write a little traffic application and use transponders to regulate these planes? You got pilots out there taxiing around runways like Alzheimer's patients in a shopping center parking lot.

What about that plane that crashed because they took off from a closed runway and crashed into a wall because the guy in the tower was too busy to notice? It comes down to a guy in a tower because the FAA cannot write software to save its life – just like the FBI they gave up on their new system and just kept using the old one, guys in towers yelling “Stop! Stop!” as they see one plane starting to taxi across a runway another plane is landing on.

How we process information matters. You'll hear later about an enterprise system I developed in Lisp to better manage clinical trials. Too bad neither we nor IBM could convince anyone that they should use our system. Had we succeeded medicines would be developed faster, more safely, and at lower cost allowing drug companies to produce more medicine and charge less and still make good money.

Information and how we process it matters. The Web was invented by Tim Berners-Lee and it sat originally atop an infrastructure paid for by the killing industry, the Defense Department. Moral: quality software makes good things out of evil things.

The Internet makes us all more efficient at everything, reduces travel, and probably will play a large part in making Earth civilization run greener. Funny how that works.

Information matters. Almost every story in the US about some governmental evil being corrected comes down to the same thing. You do not know this if you do not read the whole article because the punch line invariably appears in the last paragraph and it is always the same last paragraph: “The information that brought this matter to light was obtained under The Freedom of Information Act, also known as the “sunshine” law.”

If information matters and Lisp is the best way to process information and you know it and you are not doing something to advance Lisp then you have some work to do. The title “Why You Should Not Be Here” has to do with this being a Lisp conference. Lisp is fine, we do not need to talk about it. If you have questions about Lisp, please see the hyperspec or ask on comp.lang.lisp. But don't go to a conference about Lisp, fer chrissake. Go to a conference on the climate or meat production or poverty or militarism or even just on commerce because even a better business makes the world a better place.

If you have a Formula I race car, do not spend all day in your garage polishing it and for the love of God do not take the bus to a conference on automobile transmissions. Or high-speed windshield wipers.



Go driving.



If you have a Chinook helicopter in your backyard, use it to airlift food or medicine to those who need or pick Katrina victims off their roofs, don't drive to a conference on air traffic control. God I love analogies.

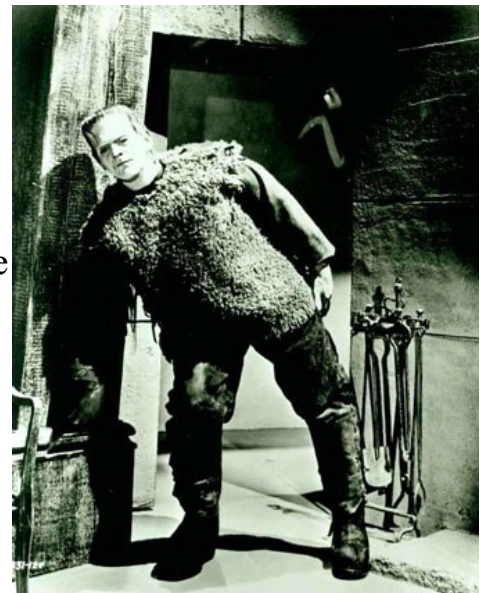


Now it turns out I was wrong -- get out your camcorders, I may never be wrong again -- but it turns out Arthur and Edi have done a great job and put together a conference after my own heart, a conference about building applications with Lisp. Yeah!!!! So if you are not writing a Lisp application now or working on an open source Lisp library, maybe this conference will help the light go on. Then get to work on something.

Or you can start using Lisp at work no matter what they say for your one-off tasks and for prototyping slash proof-of-concept. You also need to be a pleasant drumbeat of encouragement to use Lisp. If you are worried about what people will think, remember that you are right and they are wrong and as an engineer you have an obligation to the people signing your paychecks to give them your best advice. Lisp.

Unfortunately most Lispniks are just walking dead, rubbing elbows with working Lispniks on c.l.lisp or at conferences to puff themselves up while never writing a line of code themselves, a bunch of burnouts.

Almost as bad are the #lisp IRC yobbos working on peephole optimizers for SBCL. Talk about fiddling while Rome burns. We have plenty of fine commercial and even free Lisps out there, use them!



Here is a handy comparison chart to help you understand the uselessness of SBCL tinkering. Yobbos:



Kenny:



Yobbo:



Kenny:



Hope that helps.

Is it strange to hear a computer language described as a world-saving, morally-bound ethical social do-gooder issue? Only if you have never stopped to think about how important is information or if you do not fully understand the superiority of Lisp.

The good news is that Lisp continues to grow and noobies like Frank Goeninger and Andy Chambers and Peter Hildebrandt and even Kenny are Actually Working on code.

Yeah, I am a post-winter, post ansi-standard, Lisp noob. I found Lisp and started doing things like portable GUIs and a truly universal FFI and almost a binding generator if one of you would like to help finish Verrazano and now I am helping Andy Chambers push Rails and Ruby into the sea with something called OpenAIR....why am I so wonderful? Because I am a simple application building noob who sees Lisp as just a tool.

Most of you people have this incredible tool kit and are doing nothing with it because so many of you are academics. Academics are in such a panic about producing brilliant papers that they have no time for applications, because applications are really hard and that is why an applications guy (moi) developed Cells and an academic like Guy Steele took constraints and created a dataflow winter by cocking it up so badly. But I digress.

This is a Lisp applications conference and I plan to honor that by talking about applications and libraries you can use to build them and not about Cells. Cool, I never got a standing ovation before.

Well, OK, just three minutes for anyone who absolutely has no clue what Cells is about. Perhaps some yobbo could check me on his wristwatch and make sure I do not go over?

Learn Cells In Three Minutes

The code:

```
(defclass world ()
  (heard :initform (c-input () nil) :accessor heard)
  (said :accessor said
        :initform (c-formula ()
                    (bwhen (h (heard self))
                        (cond
                          ((string-equal h "Hello") "Hi")
                          ((string-equal h "Good-bye") "Seeya")
                          (t "Come again?"))))))
  (defobserver said ((self world) new-value prior-value prior-value-bound? cell)
    (when new-value
      (format t "~&You say ~s, I say ~s." (heard self) new-value)))
```

Evaluate:

```
(defparameter *w* (make-instance 'world))
(setf (heard *w*) "Hello")
-> You say "Hello", I say "Hi".
(setf (heard *w*) "I am outtahere")
-> You say "I am outtahere", I say "Come again?".
(setf (heard *w*) "Good-bye")
-> You say "Good-bye", I say "Seeya".
```

1. Changing what is heard automatically changes what is said, as determined by a declarative rule for the said attribute that does no more than access what is heard thru a normal reader to establish the linkage.
2. To allow the consequent model to do something useful other than go thru a bunch of internal state changes, observers get a crack at processing every change. Of course we are usually in an event loop or reading a stream and nothing stops us

from kicking off procedural code that simply reads the model and likewise performs output, an example being receiving an update event and then traversing a model of view instances to render them to a GrafPort, if you remember your Mac OS and Quickdraw.

That was not, too bad, was it? OK, now onto Lisp libraries and applications I think you should know about. By the way, if along the way you see an open-source project that looks like fun and you think you have something to contribute, guess what I want you to get up off of and do if you are not now doing anything to move Lisp forward and help the world.

OpenAIR: Web Programming With Cells Inside™

Ok, let's start with the newest and most exciting thing I am involved with, OpenAIR. The name is a play on Adobe AIR which can convert rich Internet applications based on Flash into desktop applications at the push of a button. The desktop is not *really* an objective of the author, Andy Chambers – we just want to make Web programming way cool, easy, and powerful – but when I brought up AIR Andy said we might be able to do something clever with WebKit to generate desktop apps as well. WebKit as I understand it is the innards of Apple's Safari web browser.

So why will OpenAIR be so cool? First, Andy picked a solid base, JQuery and cl-who.

Next, developers will trivially author dynamic, highly interactive Web pages with a minimum of coding, using declarative rules by which they can have one thing depend on another, all with transparent, natural Lisp syntax. Don't ask me how that will work because I promised not to talk about Cells.

Second -- and I do not understand why this is a big deal, maybe it is true of any Lisp tool using cl-who -- but Andy (who is new to Lisp but not web programming) says we already have a leg up on other xhtml authoring environments in assuring one's xhtml is sound.

Third, because the overall logic of building the page will be divided up into all these declarative little rules by Cells-style programming there is a good chance many of them will not need server resources and we will be able to selectively and automatically promote them to run on the client. Does that make sense?

If the whole page is one rule it will definitely need server resources. If the page is split into two rules, there is still a good chance some bit of each half will hit the server. Only when the page is decomposed into dozens of rules can we expect to find rules that happen not to reach out to the server.

For the dataflow model to carry over to the client we just need to make a Javascript version of Cells and then use Parenscript to translate the body of the rule and ship whatever does not use server resources over to the client.

Wouldn't that be sick? An RIA written on the server that automatically dynamically selectively moves as much as possible of itself to the client? The really neat thing would be avoiding having even to declare that some bit of information is on the server, if again the programmer can just write normal code and the underlying engine can identify on its own what does or does not require a trip back to the server. But that would be gravy.

By the way, imagine a rule with a conditional expression where only one branch requires a server resource. Then as the condition varies from true to false depending on other changes the rule will be moving back and forth between client and server!

I love this game.

Oh, I forgot one thing about OpenAIR: the xhtml generated is not a monolithic block. Each node will be assigned a unique key. The expansion of the Lisp macrology of a container will “include” a sub-tree of xhtml using an extension to xhtml we are calling “xlookup”, looking up the unique ID of the child in a client-side Javascript association list.

Can you say object identity? Sher ya can. And how about dynamic? Yep. We will be able to update the exact logical minimum of nodes on the client and have the Right Thing displayed JIT, just as we can modify Lisp functions at runtime and have the new version run the next time through.

Let me close the discussion of OpenAIR by pointing out that it looks easy, it will kill Rails, and it will do for Lisp more than Rails did for Ruby which is a lot. If you want to help out, Andy has just applied for a common-lisp.net hosting. Keep an eye out for that or look for us on comp.lang.lisp.

Triple-Cells: RDF Triple-stores with Cells Inside™

The thing I am second-most excited about is going to get me in trouble with Arthur and Edi because they said the talk should be about Lisp and this not only about Lisp. But I think I found a clever loophole.

RDF triple stores are not necessarily Lisp, tho Franz offers a nice one called AllegroGraph, but they are Lispy, and that is exactly why I am excited about RDF.

Hmmm. I get in trouble anyway with RDF because everybody knows they are for the semantic Web, not general-purpose persistence. Buddha told us 2500 years ago that the root of all evil is discrimination, and he did not mean just racial discrimination, he meant any discrimination. Never decide what things are, just let them be them. That may not be English.

Anyway, excuse me if I take the capabilities defined into RDF and do what I like with them. With RDF Triples we store all data using three attributes, each of them indexed sequentially: subject, predicate, and object. That is a lie, a fourth attribute called the

named graph seems to be a standard extension now, it too indexed. And Franz's AllegroGraph goes one further and indexes the sequential numeric ID of a so-called triple.

They are called subject, predicate, etc but I played with AllegroGraph for a while and pretty much stored what I wanted where I wanted and it all worked. That might not be something one wants to do too much, I guess you just quickly lose your mind if you totally abandon the idea of a sentence.

So why am I so excited? Freedom. No relational Schema to define up front, no third normal form to worry about (not that it is a bad idea), no all-tables-all-the-time, no DBA to ask for permission, and not even an object class hierarchy one would need with my former love, the persistent object database. If I get to some point in my code where I want to write out some new data I just make up a predicate and keep going? The Lisp Way, don't fence me in. Give me a powerful tool, tell me how it works, now go away.

A perfect example of why I love RDF came up when I did a lite version of Cells for RDF. That is right, persistent Cells. Change the database over here and other changes propagate throughout the persistent store. How scary is that? And if your GUI reads the database to display something and the database changes, your GUI will update by itself. Stand back, Argentina!

Oh, sorry, I am not supposed to be talking about Cells, you went to all that trouble to stand up and applaud when I said I would not. So here is the perfect example:

To make Cells work I need a two-way link. The reading Cell keeps a list of the Cells it read and each Cell keeps a list of the Cells that read it. Trust me, it is necessary. Now I maintain these with simple Lisp lists in slots of in-memory Cells and we do not want these to get out of synch so I have a whole little linking and unlinking API.

When it came to RDFs I realized, omigod!:

```
(3c-write <reader> !3c:uses <cell read>)
```

I just invented the “uses” keyword on the fly and I was done! Because all attributes are indexed it is trivial to find my dependencies or dependents:

```
(3c-read :s me :p !3c:uses) or (3c-read :p !3c:uses :o me)
```

Does this introduce uncontrolled database chaos? On the contrary, we end up with no more infrastructure than is logically needed by the problem. The solution is perfect, and the feeling of freedom and power unmistakably Lispy.

By the way, don't panic all you free software-using, food-stamp eating, windshield-washing, thrift-shopping tightwads, there is a free RDF project called Redland you can struggle with for hours on end to give you an excuse for not producing anything of value to the rest of the world or even yourself.

Clinical Drug Trial Management

OK, I apologize, RDF was really low-level and we are supposed to be talking about applications. Let's do super hi-level and something I did from 1999 to 2003, a beautiful application that was a technical and functional success but failed (for now) only because we were too small and IBM is not as smart as their commercials make them out to be, not even at selling. But a lot of people in the pharmaceutical industry who know about clinical drug trial management thought our software rocked so if you are looking for a huge opportunity get in touch. I digress.

In a nutshell, it was the mother of all EDC systems, a.k.a. electronic data capture. They like electronic because even now the documentation to support a new drug application can arrive at the FDA in a tractor-trailer, there is a bit of it by the time one is done with a big drug trial.

The business plan called for managing hundreds of clinical trials a year. Tough job: every clinical trial is different, involving a different set of forms for collecting patient data and different business rules of arbitrary complexity for each document, page, subsection, and field.

The forms themselves could be anywhere from a few pages to more than a hundred. They would change during the run-up to the trial as review boards required changes to the study and then they could change once a trial started as the drug company decided to modify the trial in light of experience. This was allowed by the FDA and possible because patients were enrolled over a stretch of time, we just had to keep track of which patients participated in which variant of the overall trial.

Another requirement was that remote clinical sites be able to use the system even if the network was not available. So no client-server and we have to replicate the database out to the clinic and keep the clinic databases and central database in synch with a homebrewed partial replication scheme.

We also had to scan these hand-entered forms back in; dynamically tell recognition software where to find what kinds of input fields so they could read the data; and then allow clinic personnel to review the scan results and make corrections to the forms using editable WYSIWYG views of the scanned forms. Users then correct errors identified by the business rules.

Once forms were corrected by the clinical site, remote monitors needed to review them for higher order errors not expressible as programmed business rules, and to request corrections or clarifications. Ie, the application would also be a workgroup application with multiple parties working on the same documents passing data queries and responses back and forth.

The Programmable Application

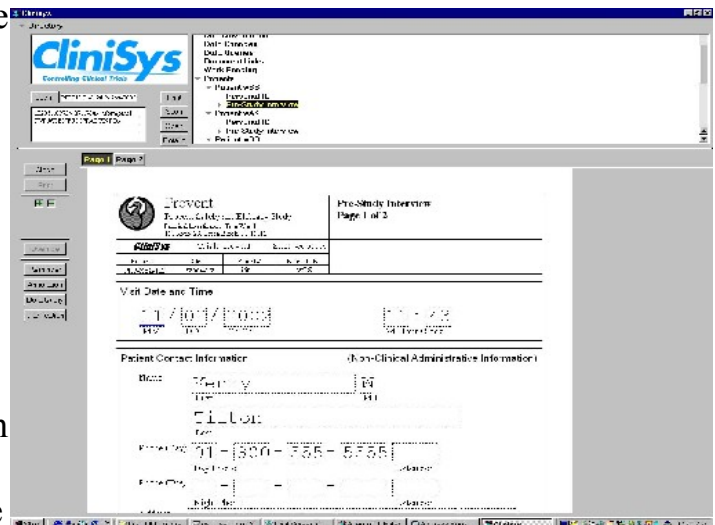
Because all the form sets and business rules were different, it was clear we needed an application configurable to a trial without HLL programming. If Lisp is the programmable programming language, we needed a programmable application.

Forms layout was easy with the usual declarative Cells magic, as was attaching arbitrary Lisp code as validation rules for any component of the document from the field all the way up to the document itself – we simply created an “errors” slot on the base class of all document classes from document down to field and wrote a Cell rule with whatever validation was required.

What was going to be interesting was getting the forms and business rules out to the clinical sites and maintaining the forms once they got there, juggling the multiple versions that would be in effect at one time.

Here too we were able to usefully animate a database using Cells. This time the database was AllegroStore, the persistent CLOS database built atop the C++ ODB ObjectStore.

It so happens AllegroStore stores class information in the DB when you store an instance. We were doing a partial replication scheme anyway, so Boom! we were done. A forms designer saved a form as a tree of objects in the database, this got replicated out to the sites handling the trial, and when they ran the application and opened a form there it was along with the business rules validating the form. Life is good.



Full marks to Franz for doing a fine job on AllegroStore (and fixing it same-day when our bizarre usage broke it) but the whole damn thing worked beautifully.

Unfortunately we were a flaky operation and even with IBM validating us and backing the business model we were not able to convince big pharma to let us in the door. The angel who had funded the effort ran out of money and IBM lost interest when they saw us lose the customer, they are like that.

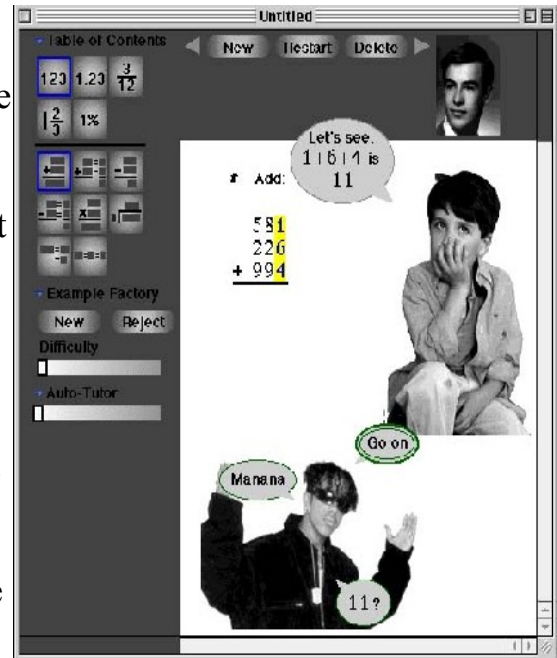
Cello: Graphical Interfaces The Lisp Way

Let's take a break from the application thing to visit another library you might like which is both Lisp and another example of what I am calling the The Lisp Way: my Cello library for building desktop GUIs. Why do I see Cello as a paradigm of The Lisp Way? Until a couple of weeks ago I apologized to folks who asked that Cello did not have

much of a widget set. Doh! Cello does not have widgets because I do not need them to create a GUI for my application. Or I would make them! Duh!

I have the core components of widgets -- base classes for abstract things like view and control, each with scads of useful configuring slots -- and I just roll them together ad hoc to create either (yes) conventional buttons or (no) a visual but widgetless way of interacting with the application. We don't need no stinkin widgets. We have some base classes and a drawing DSL and we create interface on the fly, never struggling to fit the application into the fixed mindset of a fixed widget toolbox like Tk or GTK. Of course those are fine if you want to do something vanilla and fast and if you like Cells you can use Ccltk for Tk or Cells-GTK for GTK, but my GUIs tend not to fit into the confines of standard widgets.

If you can get to my video of this talk on Google Video you can get an idea of what the interface looks like in my new educational Algebra software, and if you are interested in the software itself dig me up on comp.lang.lisp and we can talk about setting you up with a sneak preview of the software to be released in anger this September.



Ok, so you just saw Cello. Cello shares something with OpenAIR: a declarative programming model and the idea of driving a foreign framework from Lisp automatically using Cells. Actually it drives two libraries. The first is TCL/ Tk, the second is OpenGL.

OpenGL display lists work just like and in fact inspired the idea behind OpenAIR's xlookup delayed evaluation scheme: a display list has object identity. A containing display list draws subcomponents by "calling" their display lists. Display lists can be redefined at will and if you do the new appearance gets manifested in every container of that display list.

Sounds unmanageable. We have all these OpenGL nodes running around using all kinds of program state to decide what to render and when any of that state changes we need to figure out exactly which display lists to rebuild -- omigod! What are we going to do? Have every line of state-changing code worry about flagging one or more display lists as obsolete? Nahhh.

Display lists get built by executing the OpenGL you want to run in compile-mode, so just write your normal Lisp code to render the object based on whatever program state you like. Just do so in the rule of a Cell so if any of the state changes, the Cells engine

will see to it that the display list gets rebuilt. Isn't that special?

Getting back to the Lisp Way, I would like to propose this Cole Porter song as the Lisp anthem:

Oh, give me land, lots of land under starry skies above, Don't fence me in
Let me ride through the wide open country that I love, Don't fence me in
Let me be by myself in the evenin' breeze
And listen to the murmur of the cottonwood trees
Send me off forever but I ask you please, Don't fence me in

Just turn me loose, let me straddle my old saddle
Underneath the western skies
On my Cayuse, let me wander over yonder
Till I see the mountains rise

I want to ride to the ridge where the west commences
And gaze at the moon till I lose my senses
And I can't look at hovels and I can't stand fences
Don't fence me in

Now as much as you love the Lisp Way you may not be as crazy as me and might just want to use a standard widget set, cottonwood trees be damned.

Go ahead. As I mentioned above you can use Tcl/Tk or Gtk via Ccltk or Cells-Gtk. Another Lisp noob Peter Hildebrandt has stormed the beaches of Lisp and done an astonishing amount of work on Cells-Gtk even while at the same time tending to a minor thing like a Phd thesis. Something to do with threading, Cairo, OpenGL, and eventually Cells-ODE.

Conclusion

Well whaddya know. I managed to talk about Lisp applications and libraries for almost an hour and never mentioned Cells once.....I did? When?

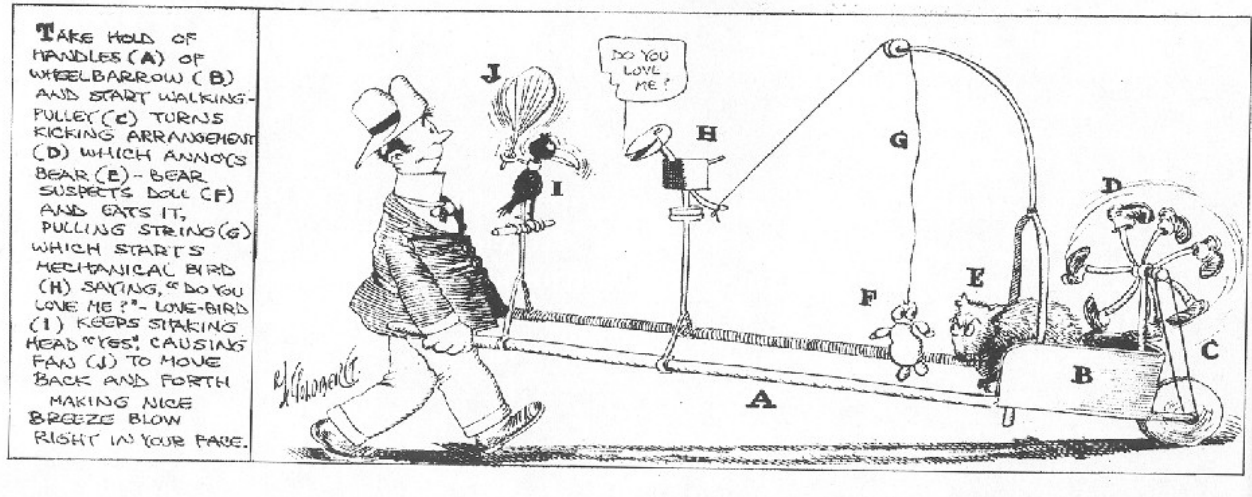
No, I was not talking about Cells. I was talking about moving information around and trying to make it as easy and reliable as possible. I was especially talking about driving one framework with another automatically, which is especially nice if we are to save the world with Lisp. You do not see the connection?

All programming should be done with Lisp, but it is not. OpenGL is not Lisp, Web browsers do not know Lisp, and neither Tcl/Tk nor Gtk know Lisp. But I can program them all with Lisp because the Cells framework lets me glue a CLOS model to any other framework out there. Cells can do so because it naturally draws us into expressing solutions with the fine, discrete granularity of the spreadsheet cell. Then with a little custom wiring Cells automatically propagates Lisp-side changes to the foreign system. A

little more code reads the event stream from the foreign framework and feeds that data back into our models. Rinse. Repeat.

Speaking of those models...

Get One of Our Patent Fans and Keep Cool



Rube Goldberg machines are held up to ridicule, but they have one quality your application models do not: they run by themselves. Mine do, I use Cells, and because I use Cells my Lisp models run by themselves and as we have just seen these models can drive any foreign framework.

Lisp world domination will not be far behind if Lispniks ever stop talking and start programming. With Cells.

Postscript

How can Cells be so cool? I have used them and bragged on them for thirteen years and only last month did I understand why. Cells is powerful because it is a library about the most fundamental abstractions, change and cause and effect.

When one cell changes it makes other cells derived from it change, too. That is cause and effect. And every change can have a callback, so programmers effortlessly propagate change outside the Cells model, as when we drive a foreign library.

Buddha said two other things. Everything is connected, and impermanence is ineluctable. Dot connection left as an exercise.