a molecular architecture for creating advanced GUIs

eric lecolinet
get/enst + cnrs/ltci

Motivations

- **New GUI toolkit architecture**

  Goals:
  - improve flexibility
  - improve source code
  - support for new Interaction + Visualization techniques

- **Ubit toolkit**
  - not devoted to a specific kind of UIs
    - general purpose toolkit
    - based on few general principles
    - advanced capabilities obtained by combining them

"things should be made as simple as possible (but not any simpler)"

Common toolkit architectures

- **Widget-based toolkits**
  - most 2D toolkits
  - properties and behaviors embedded in widget classes
  - class encapsulation & inheritance model
  - high level of granularity

- **Scene-graph model**
  - 3D toolkits (+ 2D research toolkits)
  - dynamic combination of many fine-grained objects
  - “decoration” in the instance graph
  - low level of granularity

Advantages & disadvantages

- **Widget-based toolkits**
  - most 2D toolkits
  - properties and behaviors embedded in widget classes
  - class / inheritance model
  - high level of granularity

+ **Level of abstraction:**
standardized look & feel

– **Lack of flexibility:**
stereotyped GUIs, originality -> high cost

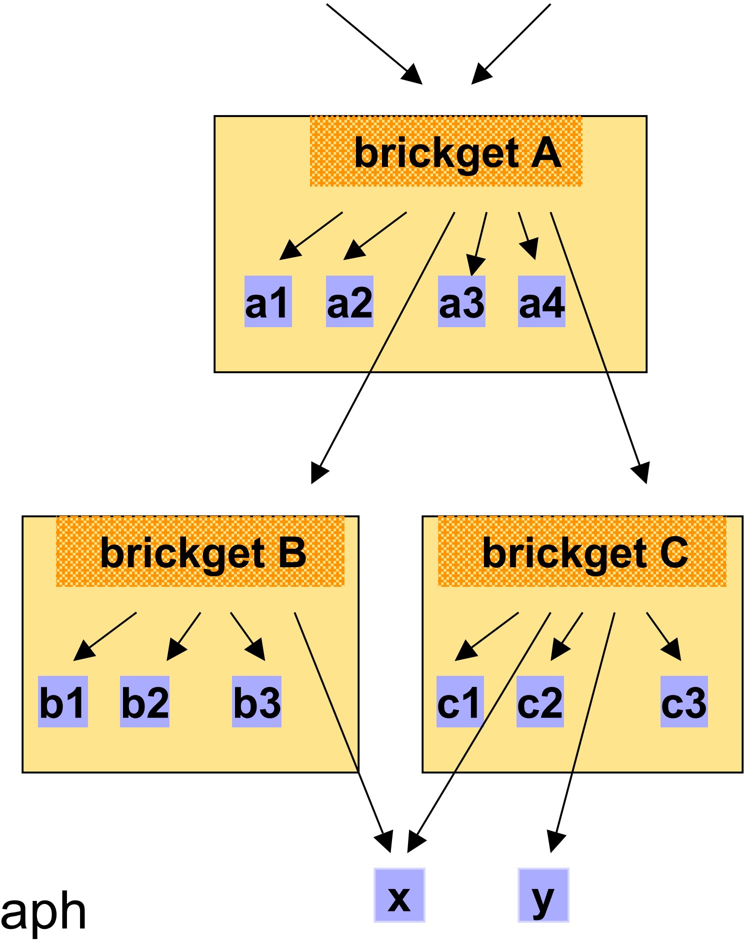
- **Scene-graph model**
  - 3D toolkits (+ research 2D toolkits)
  - dynamic combination of many fine-grained objects
  - “decoration” in the instance graph
  - low level of granularity

+ **Flexibility**

– **Level of abstraction:**
many objects,
interactors?, behaviors?

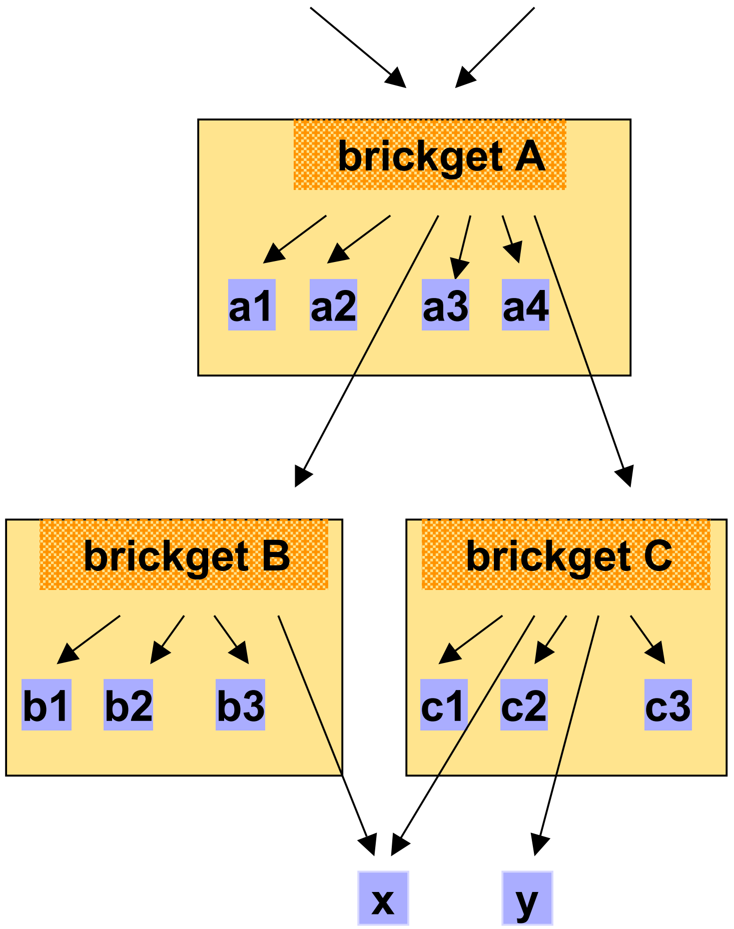
Molecular architecture

- **Hybrid model**
- **“Bricks”**
  - atomic properties and behaviors
  - reusable “services”
- **“Brickgets”**
  - mimics usual widgets
  - combinations of bricks
  - molecules, sub scene-graphs
- **Dual point of view**
  - GUI = brickget graph or brick scene-graph

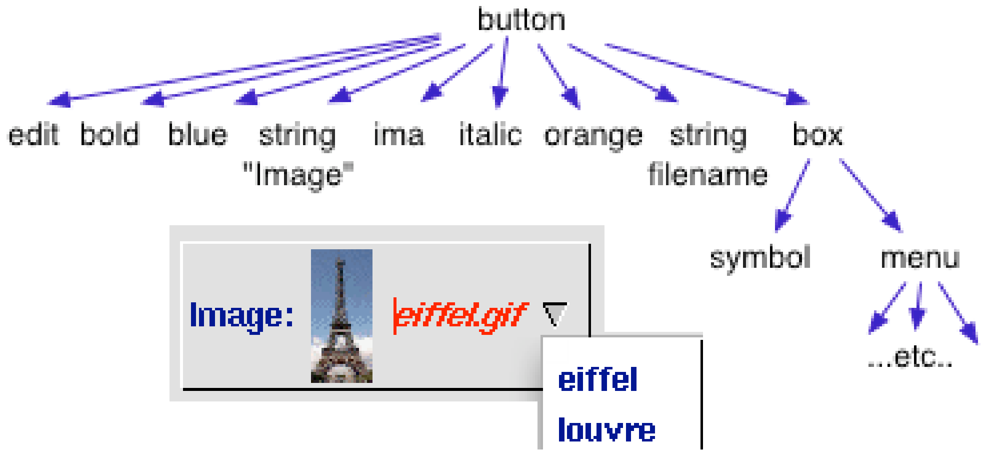


Molecular architecture (2)

- **New brickgets**
  - obtained by adding bricks
  - dynamic combination model
    - alternative to class inheritance
- **Advantages**
  - usual interactors
  - flexibility
  - reusability
    - bricks can be used in any brickget
  - configurability & synchronization
    - object sharing



Brickget model

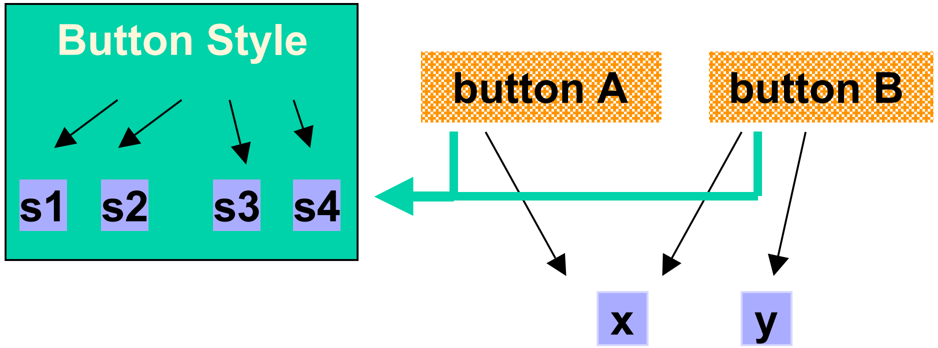


- **Brickget**
  - generic container + interaction controller
  - standard brickgets: no “attributes”

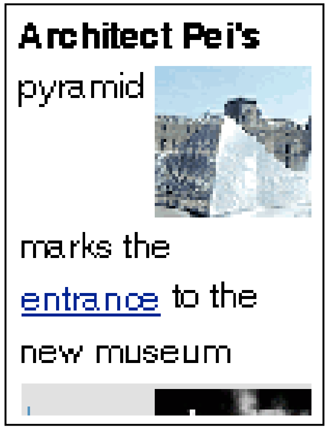
- **Brickgets attributes**
  - default values specified in bricgket class **Styles**
  - **inherited** in the instance graph
  - **explicitly** added as children

Styles and inheritance

- **Style**
  - collection of bricks
  - shared by class instances
  - context-dependent



- **Inheritance in the scene graph**
  - specified in Styles
  - powerful parameterization technique
    - very few attributes need to be specified
    - propagation of "conditional flags"

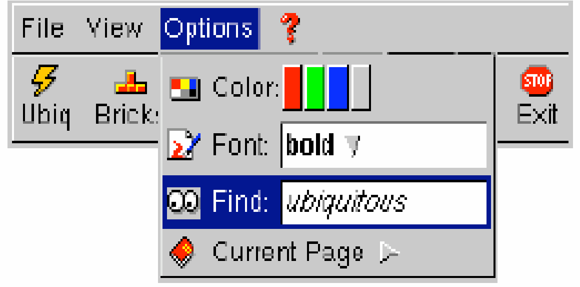


Architect Pei's
pyramid

marks the
entrance to the
new museum

Atomic bricks

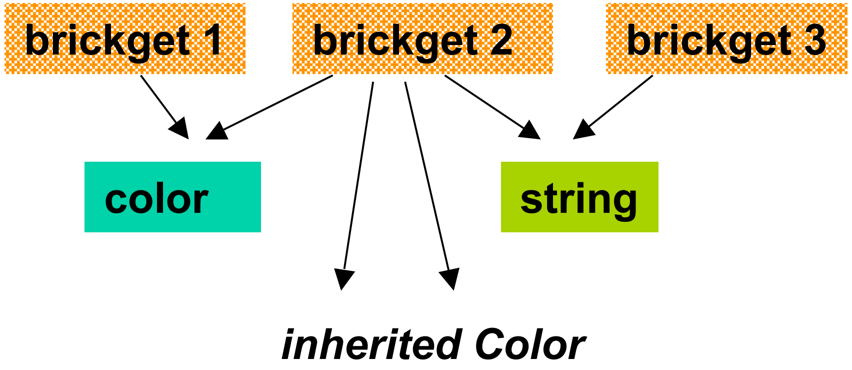
- **Bricks**
  - viewable elements: *strings, images, graphical symbols...*
  - graphical properties: *colors, fonts, decorations, scale, alpha blending...*
  - view renderers *automatic layout management*
  - callback objects
  - reified behaviors

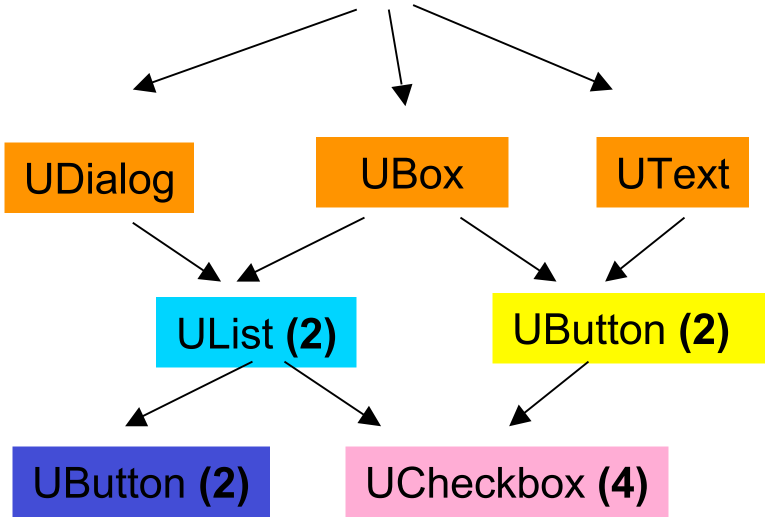


- **Behaviors**
  - can be combined
  - any interactor can contain any other
  - any standard interactor can be transformed into another one

Object sharing

- **Brick sharing**
  - synchronization, multiple views
  - configurability (+ inheritance)
  - run time memory
- **Brickget sharing: visual replication**
  - recursive: # views = # paths
  - *except for windows*
- **Molecular architecture**
  - favors object sharing
  - generalizes Swing or MVC "models”

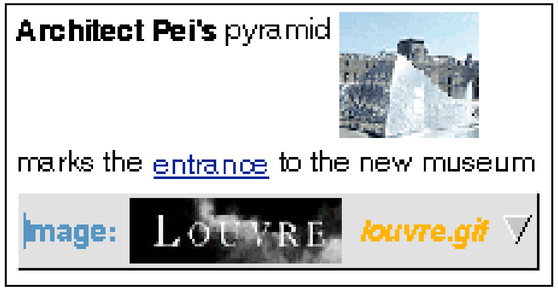




Multiple views, multiple displays

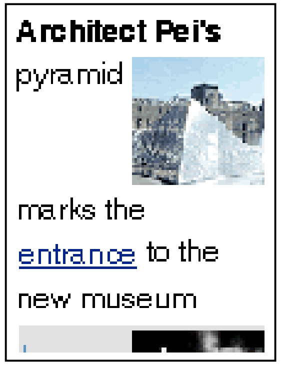
- **“Semantic” replication**

  views can differ:
  - different layout constraints
  - inheritance in the scene graph
  - conditional specifications



Architect Pei's pyramid

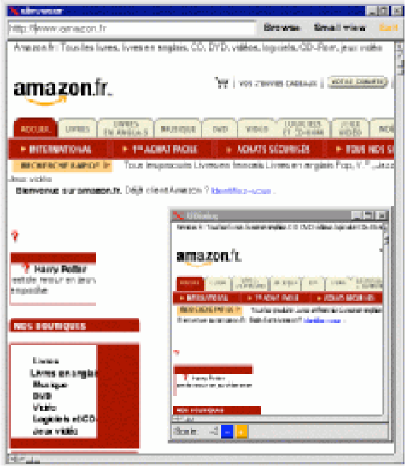
marks the entrance to the new museum



Architect Pei's pyramid

marks the entrance to the new museum

- **Remote replication**
  - 1 brickget --> N views on multiple displays
  - no restriction on the degree of sharing:
    - bricks, brickgets, subgraphs
  - “semantic” telepointers



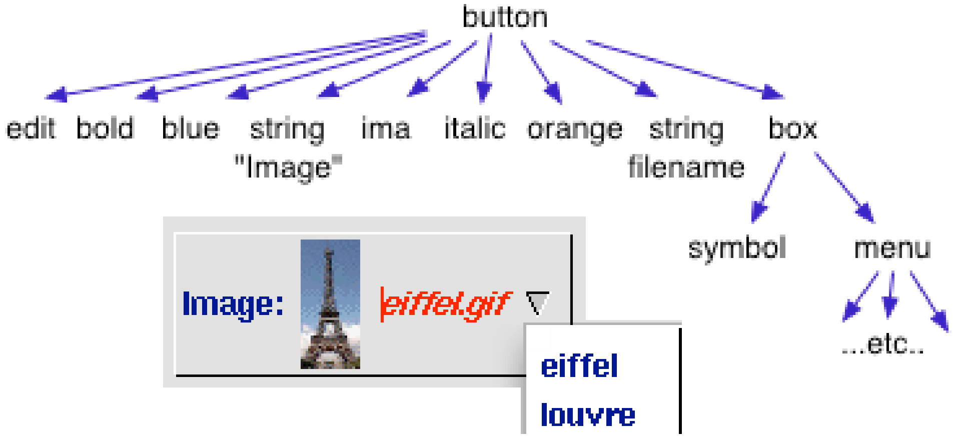
Declarative C++

- **Principle**

```
ubutton( a + b + c )
==
Button& x = *new UButton( );
 x.add(a);
 x.add(b);
 x.add(c);
```

the + operator is overloaded

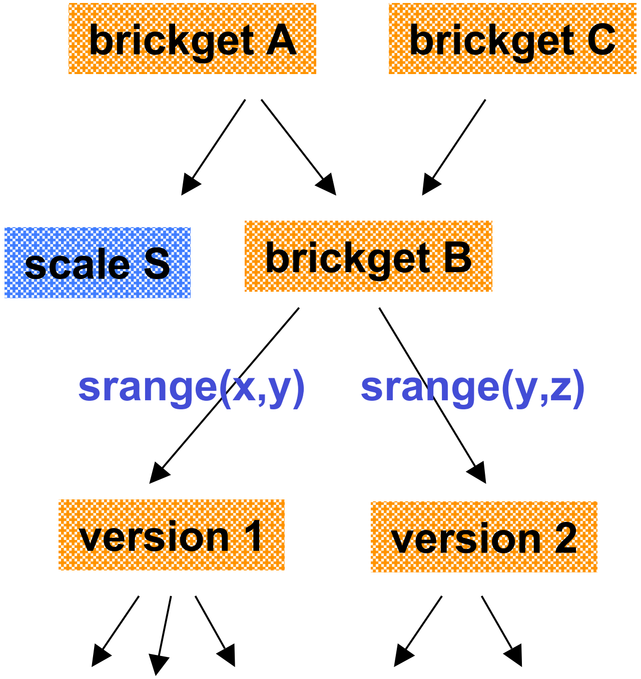
- **Compactness**
  + power of expression
  – the C++ compiler ensures syntactical correctness

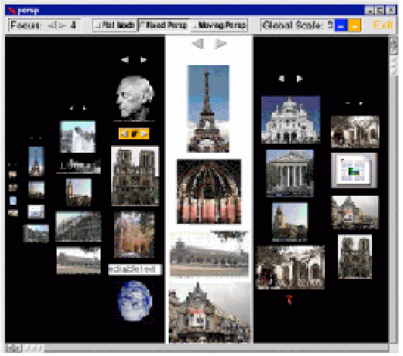


```
UBox& □example1 □= □ubutton
(
       uedit(□)
       + UFont::bold □+□UColor::blue □+□"Image"
       + uima (□"eiffel.gif"□)
       + UFont::italic □+□color□+□filename
       + ubox( USymbol::down
                 + umenu( ..etc.. )
              )
 );
```

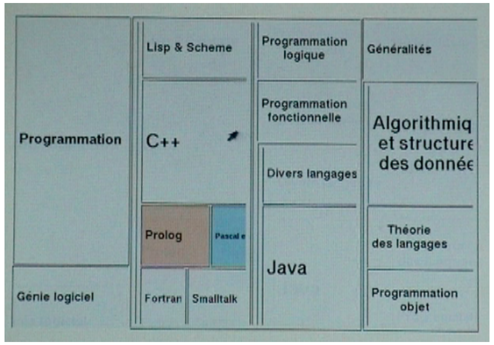
Zoomable interfaces

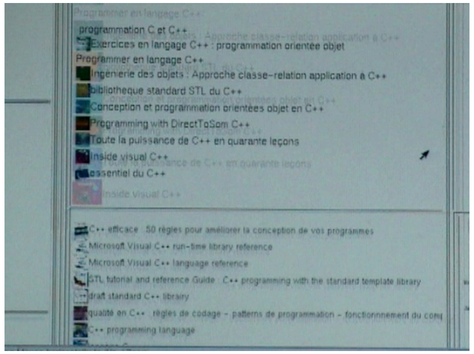
- **Combination of 2 features**
  - inherited scaling bricks
  - conditional specifications that depend on local scale





```
UBox& b = ubutton
(
      usrange(-10, -4)   /  ustr(”text")
      + usrange(-3, 3)   /  uima(”image.jpg")
      + usrange(4, 10)  /  ubox( ..... )
);
```





Programmer en langage C++

programmation C et C++

Exercices en langage C++ : programmation orientée objet

Programmer en langage C++

Ingénierie des objets : Approche classe-relation application à C++

bibliotheque standard STL du C++

Conception et programmation orientées objet en C++

Programming with DirectToSom C++

Toute la puissance de C++ en quarante leçons

Inside visual C++

essentiel du C++

Inside visual C++

C++ efficace : 50 règles pour améliorer la conception de vos programmes

Microsoft Visual C++ run-time library reference

Microsoft Visual C++ language reference

STL tutorial and reference Guide : C++ programming with the standard template library

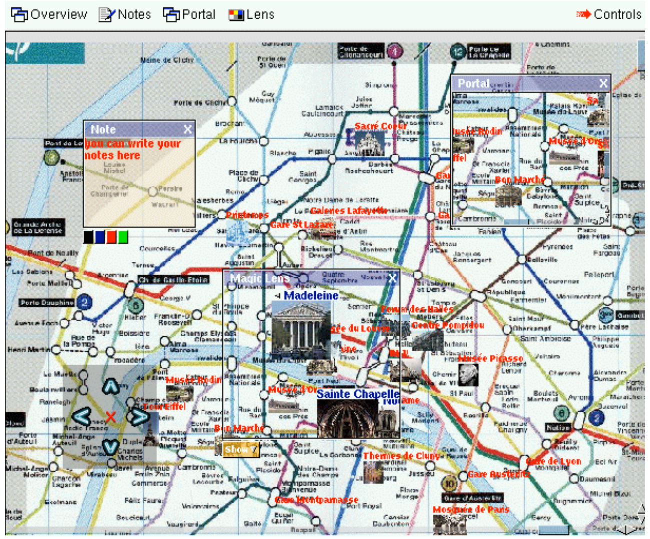
draft standard C++ library

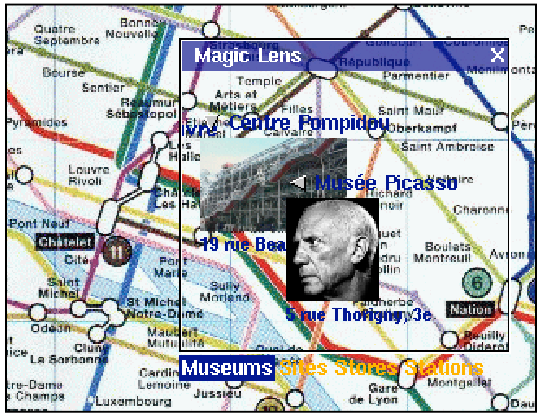
qualité en C++ : règles de codage - patterns de programmation - fonctionnement du com

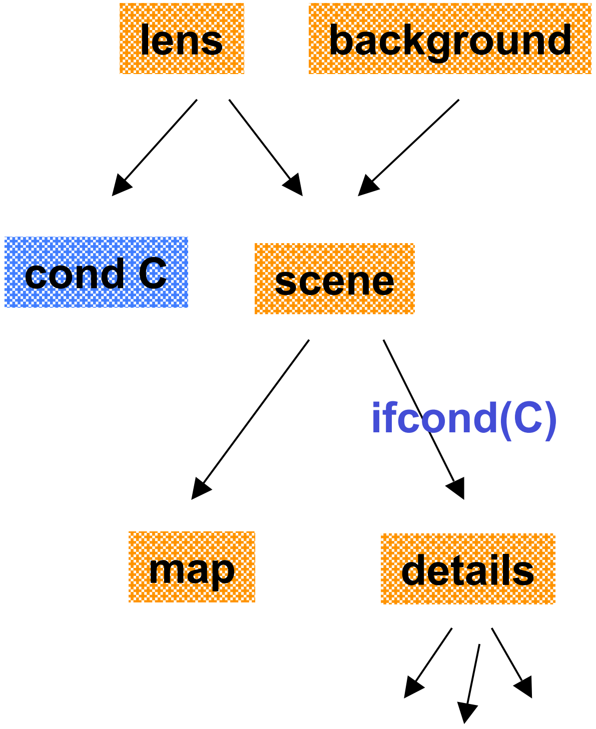
C++ programming language

Magic Lenses

- **Combination of 2 features :**
  - superimposed multiple views
  - conditional specs (inherited)



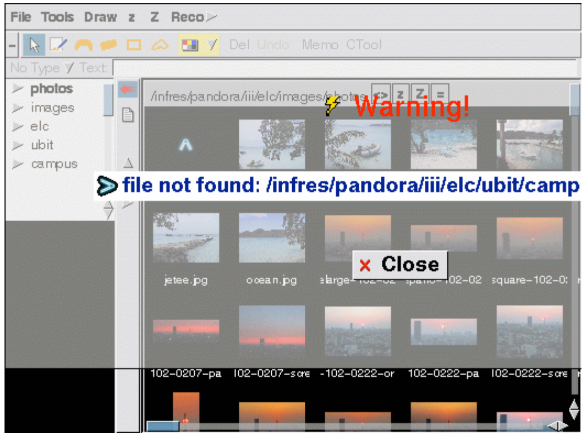




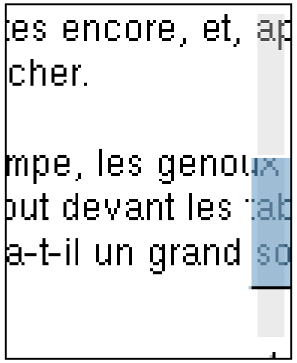
- **Can be active or passive**

Transparency

- **Transparent brickgets**
  - visually: alpha blending bricks
    - translucent dialog boxes, menus, scrollbars, Control menus
  - to events: modify or filter events
    - see-through tools

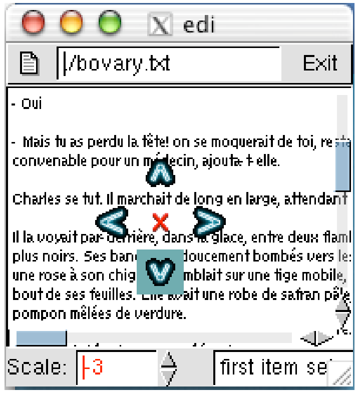


file not found: /infres/pandora/iii/elc/ubit/camp



es encore, et, ap
cher.

mpe, les genoux
out devant les tab
a-t-il un grand so

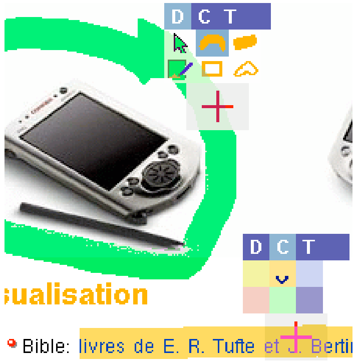


- Oui

- Mais tu as perdu la tête! on se moquerait de toi, reste convenable pour un médecin, ajouta-t-elle.

Charles se tut. Il marchait de long en large, attendant

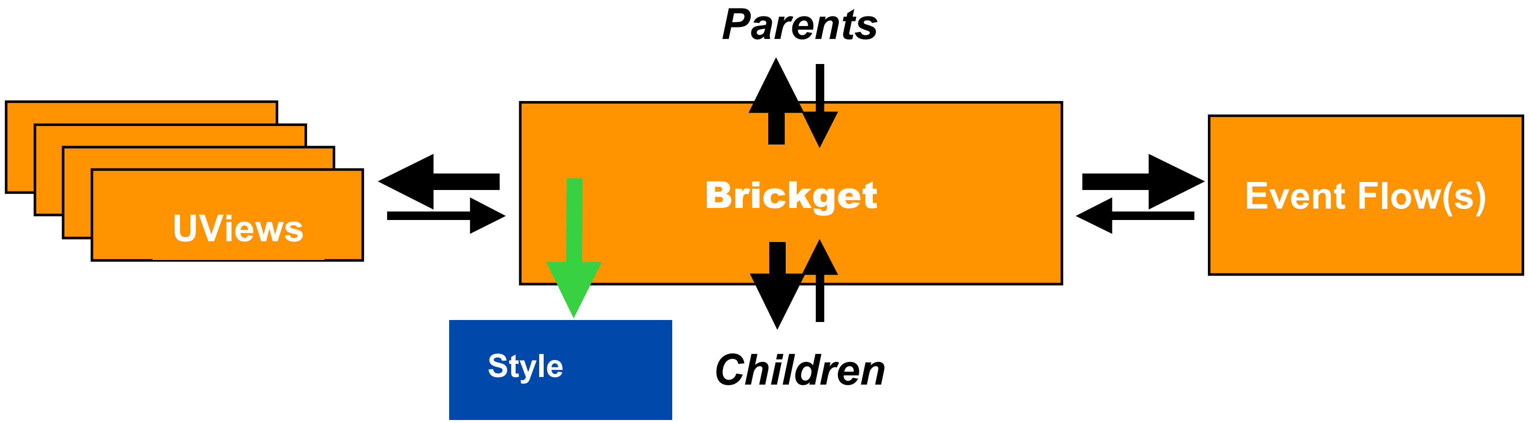
Il la voyait par derrière, dans la glace, entre deux flam plus noirs. Ses ban doucement bombés vers le une rose à son chig mblait sur une tige mobile, bout de ses feuilles. Elle avait une robe de safran pâle pompon mêlées de verdure.



Bible: livres de E. R. Tufte et J. Bertin

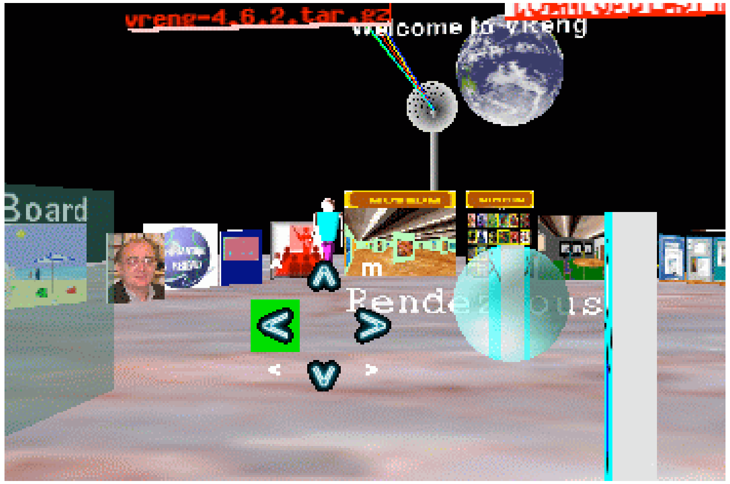
Multiple event flows

- **Bi-manual and multiple user interaction**
  - 1 or N independent event flows
  - uniquely identified



Current status

- **Current version**
  - Unix: Solaris, Linux, BSD, Mac OS X, embedded Linux (Ipaq)
  - X Window, Open GL (partial port)



- **Reasonably small**
  - 25 000 lines of C++ code
  - binary: 1.5 Mo
- **Used for various students' projects**
  - pseudo-declarative API
  - (superficial) similarity with widget-based toolkits
- **Open Source:** www.enst.fr/~elc/ubit (video)

Related work

- **Flexibility**
  - prototype/instance systems (Garnet, Amulet)
  - Ubit -> decorator pattern / safe type-checking by the compiler
- **Declarative specifications with a procedural language**
  - QOCA (constraint solving toolkit), XXL
  - new idea for encoding GUI source code
- **Object sharing**
  - Interviews, Fresco, 3D toolkits, “models” of Swing and MVC
  - Ubit -> generalization, shared interactors
- **Scene-graphs**
  - 3D toolkits and advanced 2D toolkits (Jazz, CPN2000)
  - Ubit: hybrid approach
    - combines the advantages of scene-graphs and widgets
    - unifies this approach for interactors

xxx

- novel interaction and visualization techniques rarely available
- extension by subclassing
- « static » model, high level of granularity
- most 2D programmers unfamiliar with this approach

Exemple:

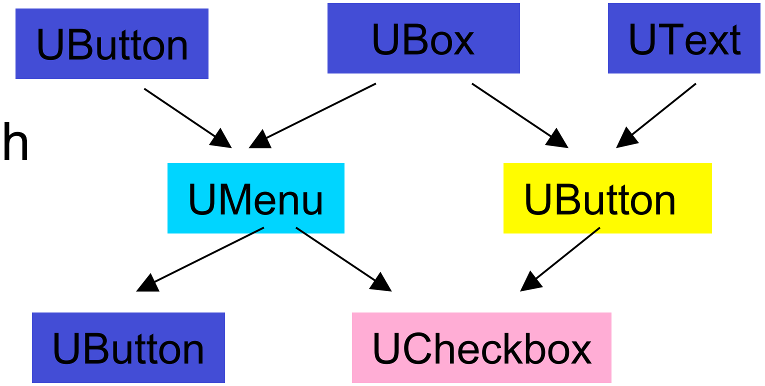
- what is a “button” in this model ?

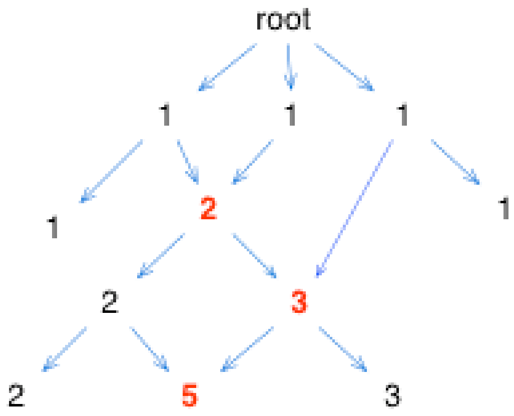
Example

- button, 2 colors, 1 font

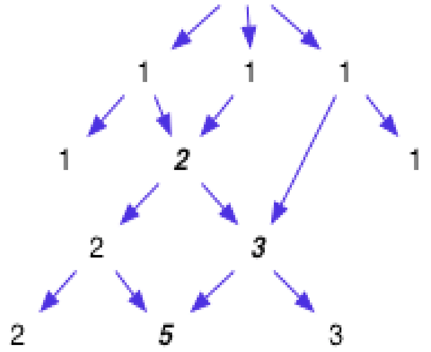
Brickget sharing

- **Interactors can be shared**
  - sharing semantics depends on brickget type (3 cases)
- **Groups**
  - intermediate nodes in the scene graph
  - appear in all parents
- **Boxes**
  - manage replicated (but synchronized) V
  - each view controls an area on the scree
- **Windows (menus, dialog boxes)**
  - not replicated
  - implicit behaviors

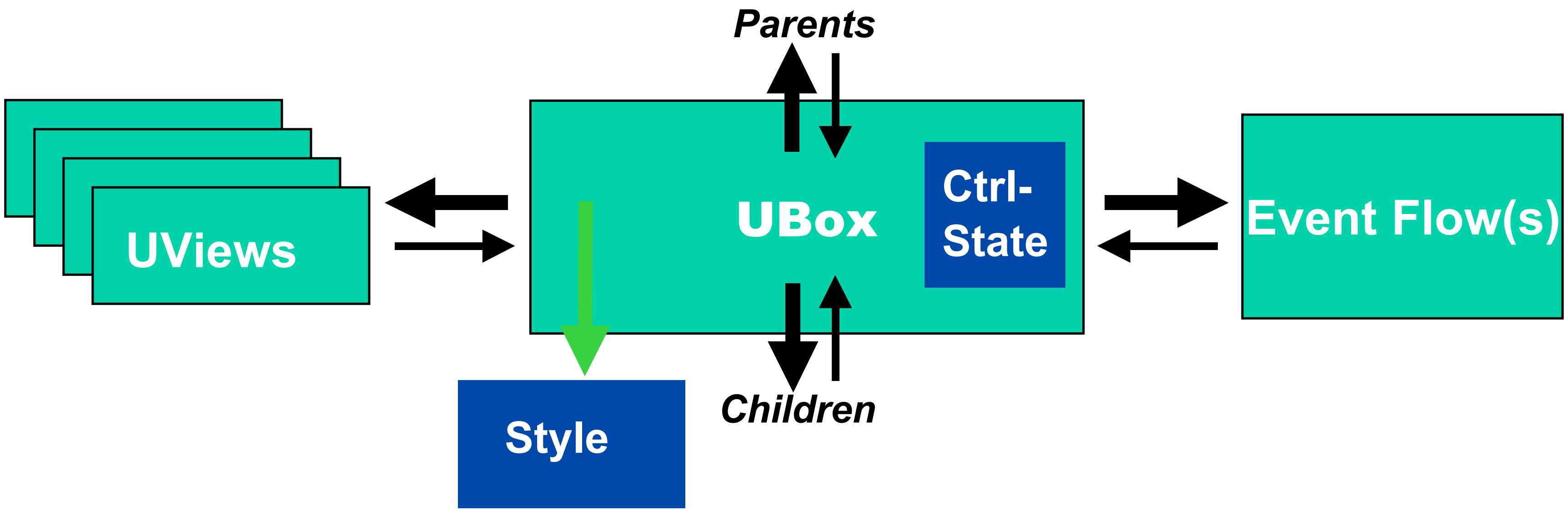




Anatomy of a brickget

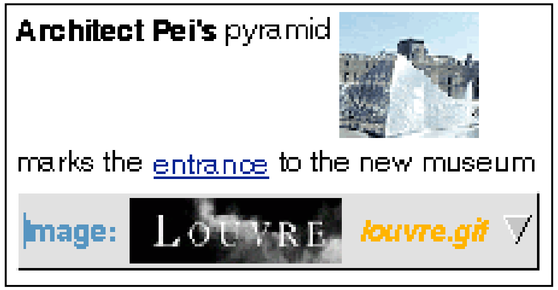


- penser aux behaviors



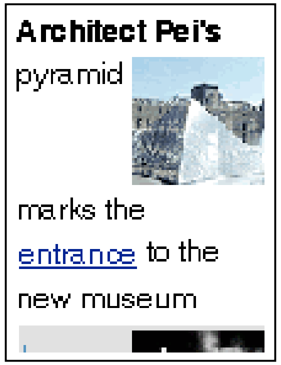
Hypertext / DOM

- **Groups vs. Boxes**
  - have no impact on layout
  - modelize “in-line”markup tags
  - no equivalent in classical 2D toolkits



Architect Pei's pyramid

marks the entrance to the new museum



Architect Pei's pyramid

marks the entrance to the new museum

```
UBox& my_page = ubox(
      UFlowView::style
       + ugroup( UFont::bold + “Architect Pei’s”
)
      + “ pyramid ” + uima(“pyramid.gif”)
      + “ marks the ”
      + ulinkbutton(“entrance”)
        + “to the new museum”
      + example1
);
```

- **Consequence**
  - representation of a HTML or XML document
  - document object model

- **constraints (Amulet, SubArtic∞**
  - Ubit -> no constrainst solvers but dependencies
  - dependencies + inheritance in the scene graph : powerful feature

- **brickge molecules are "abstarct" ?**
- **-> scene-graph can be embedded**

Multiple event flows

- **Event flow controller**
  - dispatches events to appropriate brickgets
  - only one by default

- **Multiple flows**
  - when alternate event sources are available
  - flows are logically independent
  - each flow is uniquely identified

- **Applications**
  - bi-manual interaction
  - groupware (each user controls his own pointer)
  - remote control

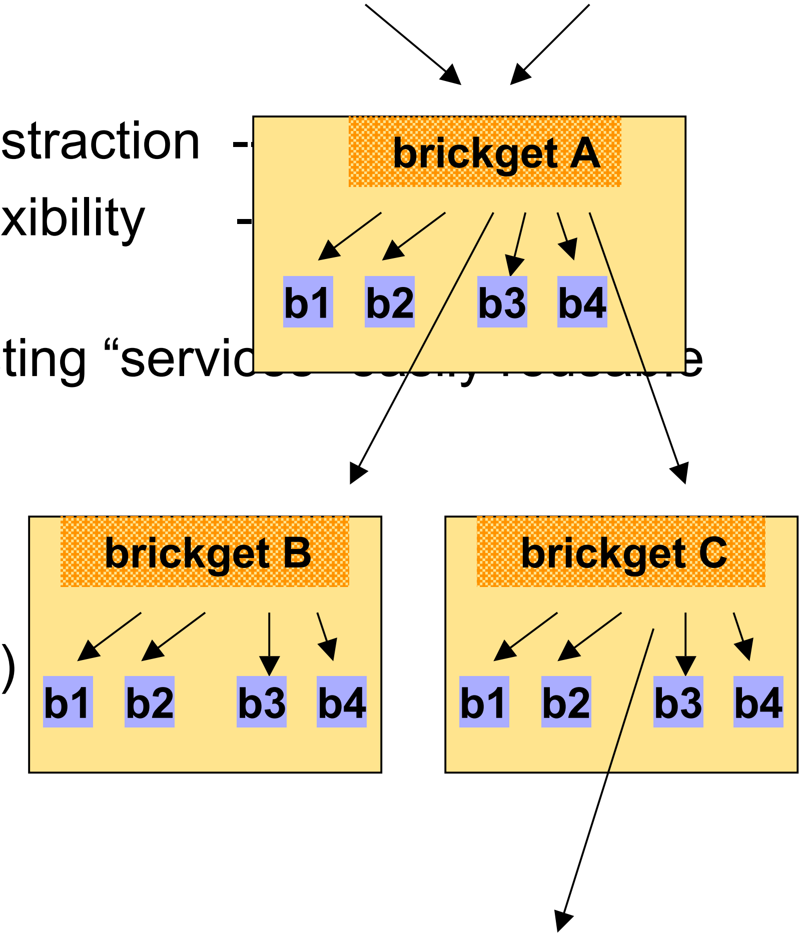
Advanced features

- **obtained by combining the standard features of the toolkit**
  - object sharing and visual replication
  - conditional specifications
  - inheritence in the scene graph
  - multiple event flows
  - etc

Abstraction vs. flexibility

- **Typical GUI needs**
  - *many* stereotyped interactors: + abstraction -
  - *some* specific interactors: + flexibility -
    & novel I+V techniques
  - *reusability* existing “servic e
  - *configurability* & *synchronization*

- **Proposed solution**
  - “molecular” architecture (**Ubit toolkit**)



- **Typical GUI needs**
  - *many* stereotyped interactors: + abstraction --> brickgets
  - *some* specific interactors & novel I+V techniques: + flexibility --> bricks
brickgets = embedded scene-graphs
  - *reusability*: bricks reusable in any brickget
  - *configurability* & *synchro.* bricks not embedded in widgets --> can be shared
- **Dynamic combination model**
  - container + decorator design patterns
  - alternative to class inheritance

Multiple views on multiple displays

- **Remote replication**
  - 1 brickget --> N views on multiple displays
  - “semantic” telepointers
- **No restriction on the degree of sharing**
  - bricks (strings, colors, images...)
  - brickgets, brickgets graphs
- **Centralized architecture**
  - advantage: simplicity
  - drawbacks: bandwidth, limited # of displays

Advantages & disadvantages

- **Widget-based toolkits**
  - + Level of abstraction
    - standardized appearance & behavior
  - – Lack of flexibility
    - classes hard to augment
    - stereotyped GUIs, "originality” is expensive

- **Scene-graphs**
  - + Flexibility
  - – Level of abstraction:
    - behaviors?, interactors?, many objects...

Declarative specifications

- **Procedural encoding**
  - large amount of syntactic sugar --> verbose, redundant
- **Declarative languages**
  - require an interpreter
  - limited interaction capabilities
- **Proposed solution**
  - pseudo-declarative C++
    - compactness, power of expression
  - object sharing
    - graph of dependencies (rather than spaghetti of callbacks)

Declarative C++

- **Lisp oriented**
  - nested lists -> object trees
  - the + operator is overloaded

- **example: composite text or whatever**
  - ubutton(arglist) == *new UButton(arglist)
  - ubutton(a + b + c) ==
  - Button& x = *new UButton(); x.add(a); x.add(b); x.add(c);

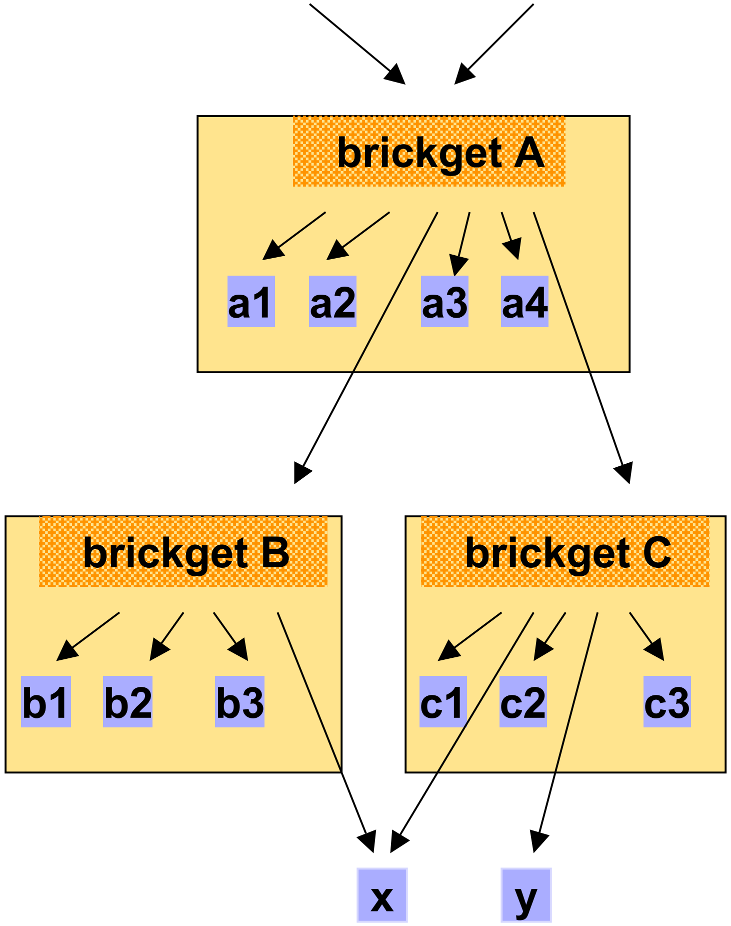
- **the C++ compiler ensures syntactical correctness**
  - simple and uniform mechanism
  - mainly based on polymorphism

Conditional specifications

- **Local conditions :**
  - example1.addlist(UOn::enter / uset(&color, UBColor::red)
  - + UOn::mpress / uima("working.gif")
- **Inherited conditions :**
  - one specification => several variants
    - UFlag f1,f2,f3;
    - b = ubutton(f1 / ustr("abcde")
    - + f2 / uima("whatever.jpg")
    - + f3 / ufilebox(f4 /...)
    -);
    - x = udialog(udefFlag(f1) + b);
    - y = umenu(udefFlag(f2) + b);
    - z = utextbox(udefFlag(f3) + udefFlag(f4) + b);
- **Generalized callbacks**

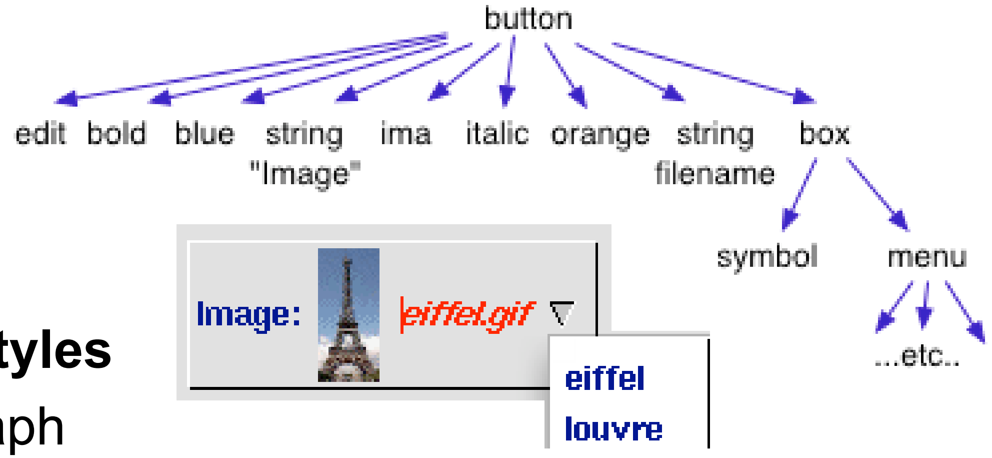
Molecular architecture (2)

- **Dynamic combination model**
  - container + decorator design patterns
  - alternative to class inheritance
- **Stereotyped interactors**
  - brickget level
- **Application-specific interactors**
  - brick level
  - brickgets = embedded scene-graphs
- **Reusability**
  - bricks reusable in any brickget
- **Configurability & synchronization**
  - brick and brickget sharing



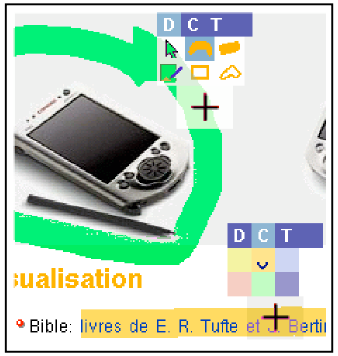
Bricks and brickgets

- **Bricks**
  - viewable elements: *text, images, graphical symbols...*
  - graphical properties: *colors, fonts, decorations, scale, alpha blending...*
  - reified behaviors, view renderers *(layout managers)*
  - callback objects
- **Brickgets**
  - generic containers
- **Brickgets attributes**
  - default values specified in **Styles**
  - **inherited** in the instance graph
  - **dynamically** added as children



Transparent tools

- **active tools:**
  - perform an operation on underneath objects
  - must have knowledge on objects
- **passive tools:**
  - event modifiers, transprent to events, no knowledge on objects
  - underneath objects may react specifically


D C T
D C T
ualisation
Bible: livres de E. R. Tufte et J. Berti