helloUML

specifications document

February, 2000

Michelle Neuringer

- Project description

motivation

- inspired by debates in cs32 TA camp -- "are we gonna make them use rose again?"
- the problem with rose: it's made for experienced team programmers and large projects -- that's its target audience
- the brown cs department teaches good object-oriented design from the beginning (cs15) and continues with the same philosophy as students progress up the cs ladder.
- what the department needs is a design tool that's **geared toward its own audience** -- fool-proof enough for beginners, powerful enough for budding hackers.

audience

- the scope of the cs department
- *beginning programmers grappling with design issues (CS15, 16)*
  - object decomposition
  - visualization of object relationships
  - learning how code follows design
  - presentation of design to TAs
- *beginning programming teams with team design issues (CS32)*
  - module decomposition
  - team communication (deadlines, integration milestones)
  - presentation of ideas to professors, TAs, classmates
- *experienced, larger programming teams (CS190)*
  - extended use of features learned in 32
- *teams not exactly in industry (programmers working in places like consulting firms, designing more compact, short-term software solutions)*

helloUML! (standalone version)

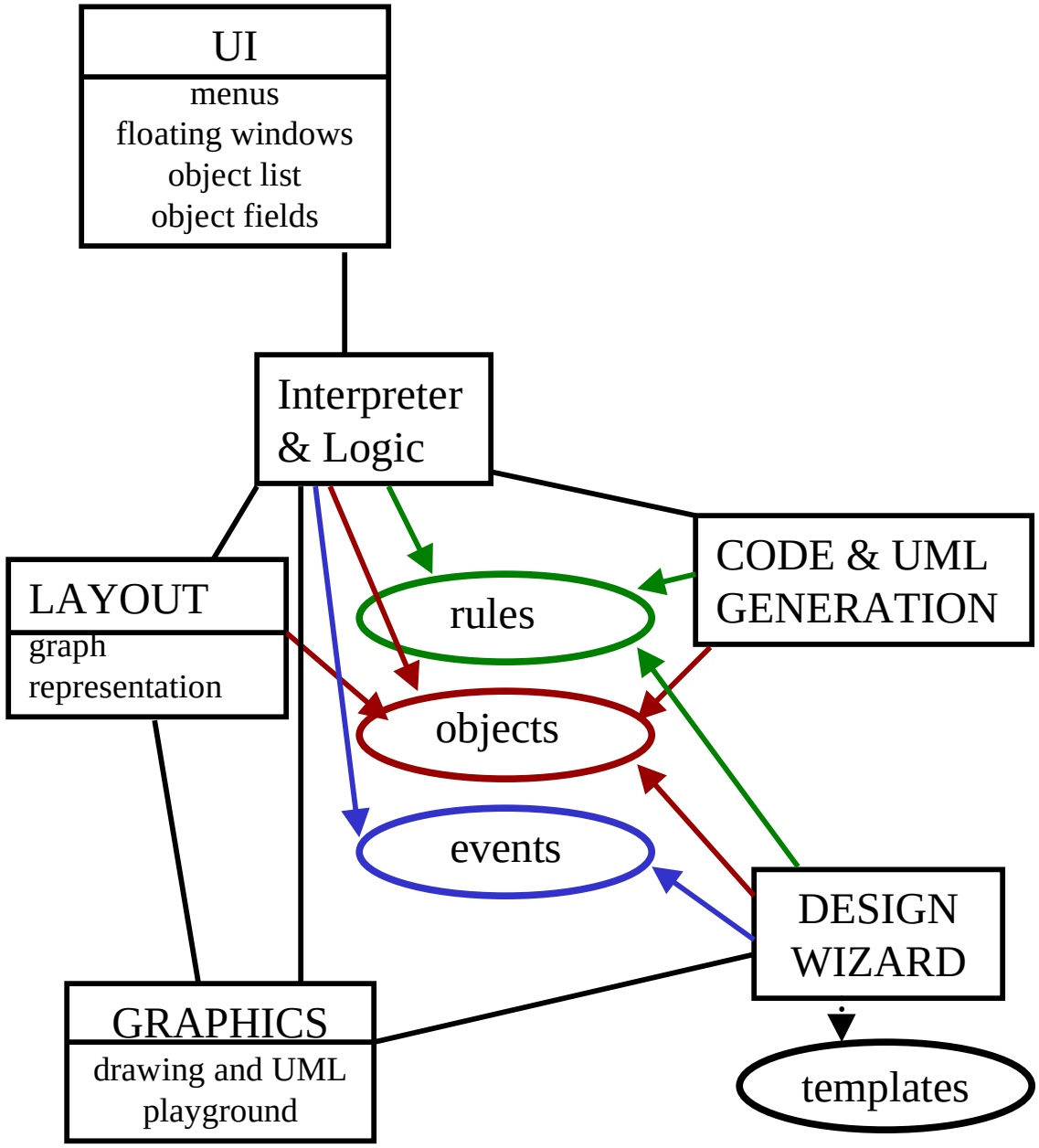
- generate UML diagrams from graphical input
  - point and click to create objects
  - objects are color-coded based on type & can be tagged & acted on (moved, associated) in logical groups
  - multiple fields for information -- yet will *never* prompt user to enter anything they don't want to
- logic for object manipulations
  - rules concerning what objects can & cannot do (based on type) & which connections are allowed &/or appropriate (based on other objects in graph)
- "design wizard" AI
  - helps user make connections between objects based on knowledge of design patterns, logical modules & program's internal rules
- automated layout for readability & presentation
  - layout algorithm organizes graph based on computational geometry techniques & its own aesthetic opinions & takes tags (user groups) into account when drawing the UML
- file & UML generation
  - generate C++ header files or .java files from UML diagrams
  - & vice versa
- allow user to select from premade design templates

helloUML (networked version)

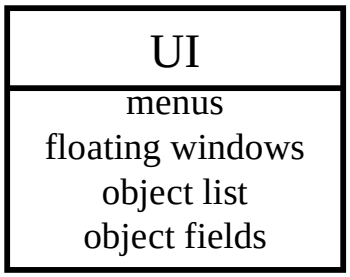
- all of the above, except all helloUML information stored in a centralized data repository
- users required to login to helloUML system
- users belong to one or more groups, similar to Unix groups

- one administrator to every group: able to give users access to the group's helloUML information for reading & writing
- additional helloUML information to be stored on the server (can be added to standalone version, but more appropriate for networked version)
  - project timeline
  - project checklist: ability to "check off" a class from a design to reflect that class is fully "implemented" & thus ready for others to safely use)
  - project member schedules

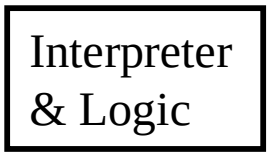
- System Model (standalone)



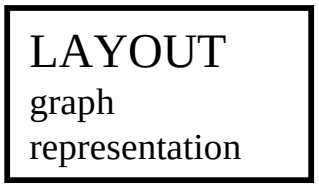
- System Model, explained



  - all (non-drawing) graphical interaction between user & program
  - separate from other modules (only connects to the UI interpreter) so that UI is replaceable



  - delegates messages from UI to necessary components, first checking whether moves are legal
  - objects are made in this module



  - layout algorithms based on graph representation (nodes & edges) of objects & connections
  - communicates with graphics & objects modules

- System model, continued

CODE & UML GENERATION

- knows rules & objects: generates code
- contains parser: generates objects

DESIGN WIZARD

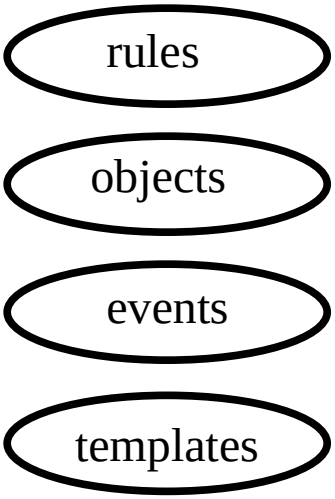
- AI uses rules, objects & user events (what moves user has made) to make suggestions about design
- can "borrow" a design (template) from the design wizard

GRAPHICS

drawing and UML playground

- implementation of drawing tools
- graph drawing (objects & connections)

- data



what the user can and cannot do

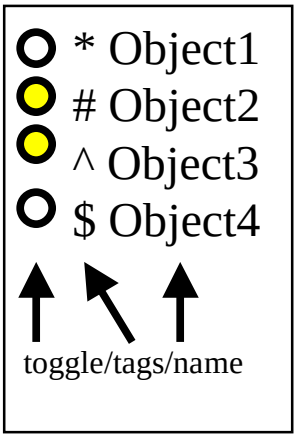
internal representation of objects in diagram

internal representation of useful & undoable actions that user performs

library of premade designs & design organizational structures

Strictly Visual: GUI & Graphics

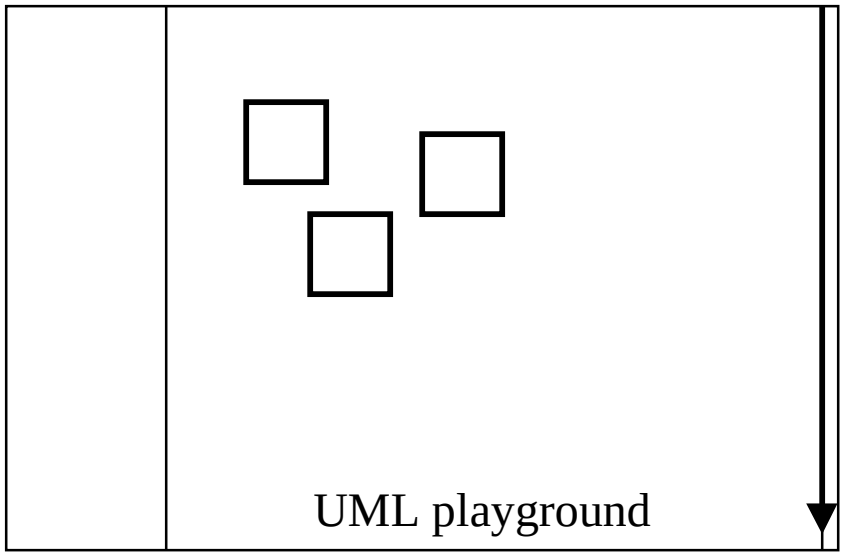
- Object list
  - where is it? left side of application, displayed vertically
  - what is it? list of all objects (hidden or shown) in UML playground
  - what does it do?
    - click on object's name in list & bring focus to corresponding object in UML playground (zoom to diagram centered around object)
    - "tag" object in list by selecting from a pop-up window of user-defined tags
      - tag shows up next to object
      - example tags: frontend, backend, ai, gui
    - show/hide object in UML playground by toggling a button next to each object in list
      - option to only view objects tagged a specific group
    - hierarchy views (lower priority)
      - inheritance
        - » example: "+ square" expands to:
          - minesquare
          - empty square
      - packages (java) & namespaces (C++)



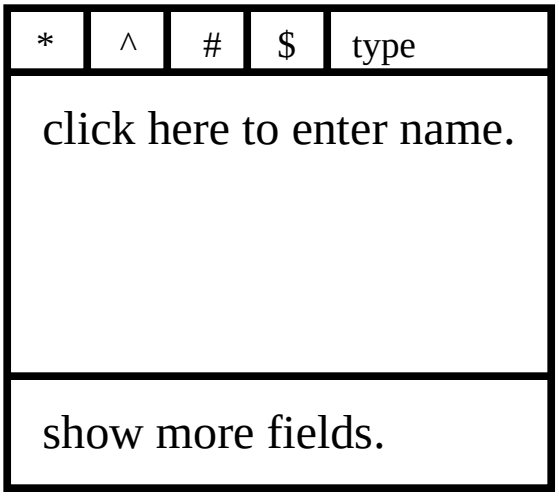
- UML playground
  - what is it? the main "canvas," where all objects "live"
  - what else?
    - control-enter inserts pagebreaks

Visuals, continued.

- pages are of variable size
- scrollable within a page as well as between pages



- Objects
  - what's an object? an object is the most basic element of a helloUML document -- a square box with properties.
  - how do i create one? Double-click anywhere inside playground to create an object. Object will appear inside the UML playground as a "blank box" with some fields...

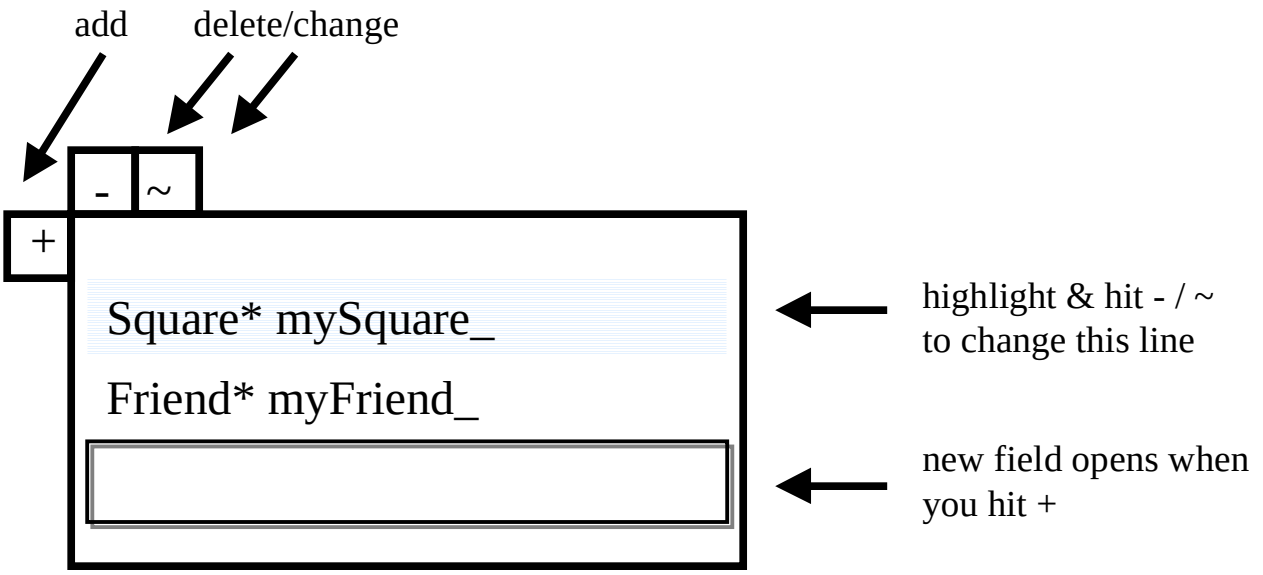


Visuals, continued.

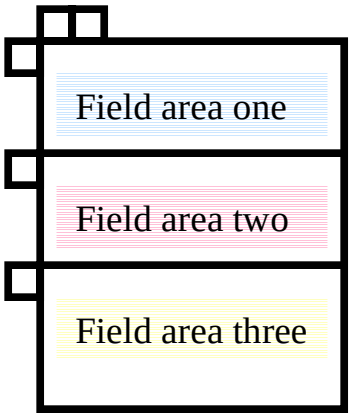
- Fields
  - all fields can be shown or hidden, unless explicitly noted
  - fields include:
    - type: represented by icon toggle buttons. Depending on the icon toggled, object will change color
      - sub: this object is a subclass. color: ?
      - sup: this object is a superclass. color:: ?
      - i: this object is an interface. color: ?
      - other/special: this object is none of the above. color: ?
      - default: color: ?
    - name: name of class. Always shown.
    - level of detail: represented by icon toggle buttons.
      - all: show all detail fields
      - var: show all variables
      - meth: show all methods
      - comm: show all comments
      - connect: show all connections (i.e., containment, association, inheritance, implements)
      - tag: show all tags (i.e., gui, ai, frontend, backend)
    - more detail fields shown == increased level of detail.

Visuals, continued.

– how detail fields will work:

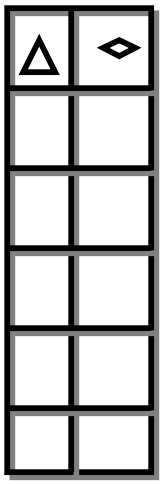


- each field area will have a (+) button. Click (+) and text field (one line) will open up. Write what you want - when you hit enter or press a button on the side of the text field, the text field turns to a label
- there will be one (-) button for all fields - this button tells the program that you want to delete all highlighted labels
- there will be one (~) button for all fields. Highlight a label from any field and click (~) - this tells the program that you want to change the highlighted label. The label will turn back into a field for modification.



Visuals, continued.

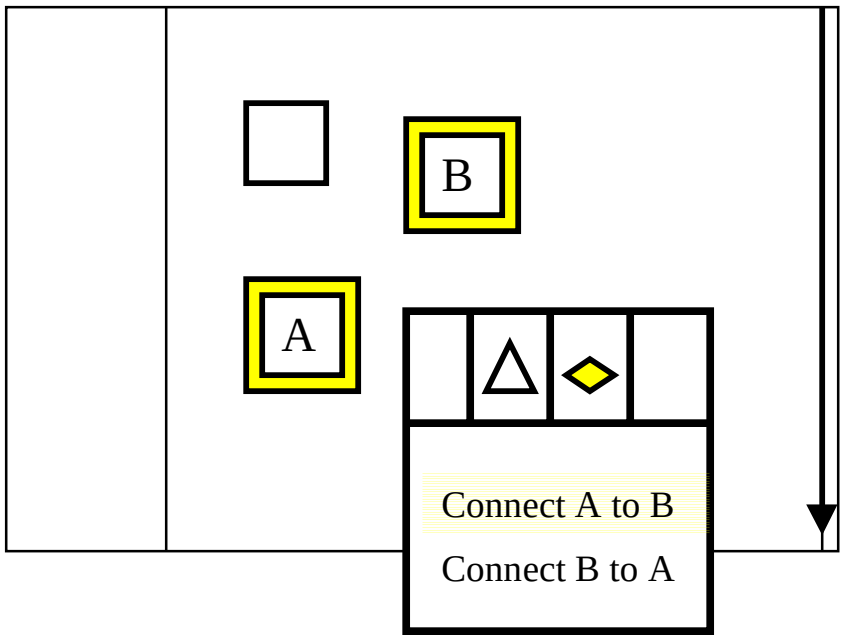
- Floating drawing menu
  - for drawing connections as well as adding external text to playground, grouping objects to move (or to apply uniform operation to)
  - contents:
    - containment diamond
    - inheritance triangle
    - association solid line w/arrow
    - implements broken line w/arrow
    - text box
    - specify line widths
    - lasso tool to group objects in order to move them
    - boxes w/color & transparency options in order to group them visually
    - import image (for screenshots)



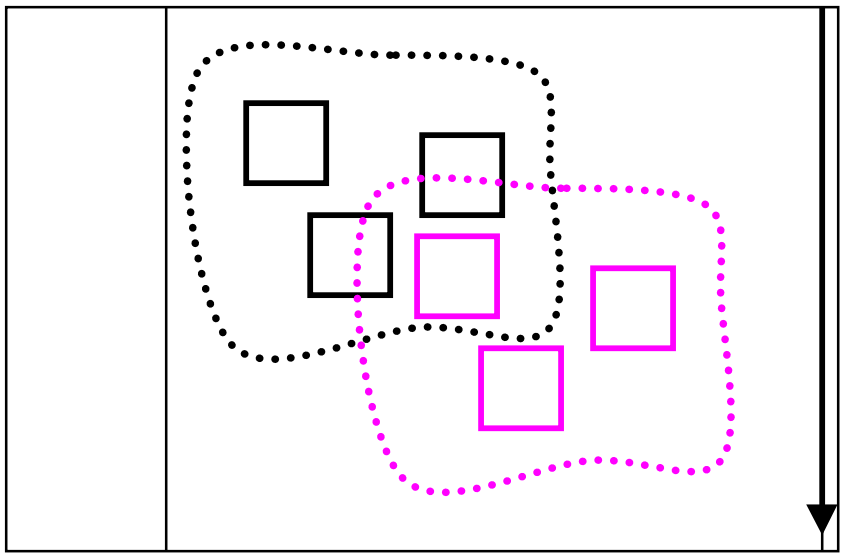
- Connections
  - different methods for different kinds of students
    - artists: use floating drawing menu
    - those who can't draw a straight line: highlight two objects in UML playground. As soon as you do this, program assumes you want to draw a connection. Box pops up with connection options (diamond, triangle, solid, broken) as well as the names of the objects you want to connect (so you can verify whether you clicked the right ones). You then specify the direction of the connection:

Visuals, continued.

– click & connect:

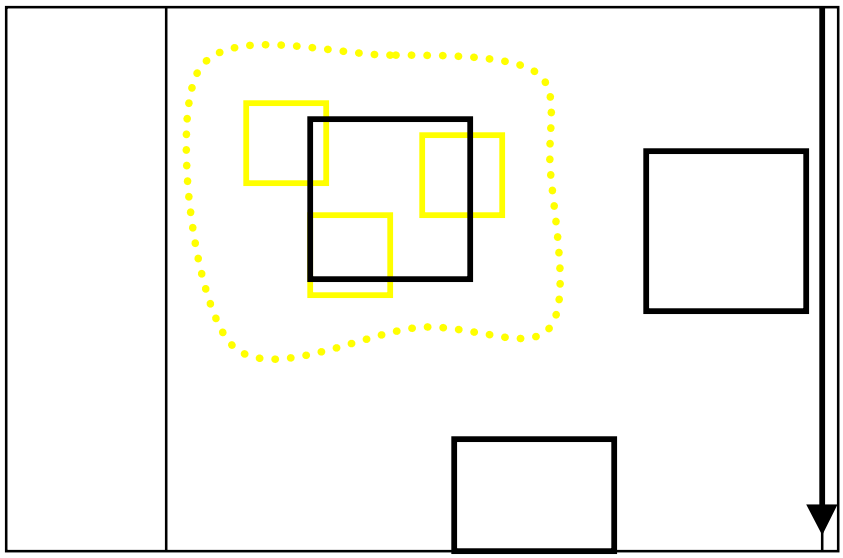


– grouping:



Visuals, continued.

- Miscellaneous
  - multiple undo: every major operation (i.e., changing an object's name is undoable, but entering a specific letter in an object's name is not)
  - zoom to <%>: zooms/unzooms the UML playground



  - preferences:
    - assign colors to specific kinds of objects (i.e., make all interfaces orange even though purple is the default color)
    - set background/foreground colors of entire GUI

- Code and UML generation
  - flow:
    - select 'generate' from pull-down menu and program will:
    - ask which language - Java or C++
    - 'design wizard' will pop up and explain that it needs to check design (make sure that all fields necessary for header generation are filled in) in order for code generation to work correctly.
    - dw will then iterate through all objects in diagram & highlight all fields that need to be filled in, checking for:
      - modifiers (private, public, protected)
      - class names
      - constructors/destructors
      - return types
      - note: option for program to put in defaults (modifiers, constructors & destructors, etc. whenever possible)
      - when all fields are filled in, code generation will begin
  - C++ header file generation:
    - print header comment, if one exists
    - print includes/fwd declares, if user decides to specify them
    - print class comment, if one exists
    - print class name {
    - print variables
    - print methods
    - print };
    - extensions:
      - » smart support for includes, fwd declares, namespaces
      - » actually generate compiling header files (no circular includes, all methods print "returning default value!!" & then return 0 or NULL)

- .java file generation
  - print package name and import statements, if they exist
  - print public class name {
  - print variables
  - print empty methods
  - print }
  - extensions:
    - » smart support for package names
    - » generate compiling classes (all methods print "returning default value!!" & then return 0 or null)

- UML generation (new since requirements)
  - parsing:
    - create UML objects (in playground/internal object structure, as if user were manually creating them)
      - » parse comments into objects' comment fields
      - » ignore code inside methods
      - » work with variable whitespace
    - base spec will not fill in connections

- Rules
  - diagram building:
    - superclass can't extend, contain, inherit from subclass
    - superclasses & subclasses can't be associated with one another
    - Java: no multiple inheritance, can only implement interfaces (can't contain, be associated with, or extend them)
    - interfaces can't have variables
    - (difficult) class must include all methods of an abstract parent in order to be instantiated & used somewhere in the design
    - warnings against large classes - lots of methods/variables

- Backend (internal representation) requirements
  - storage of objects
    - standalone: data structure of choice
    - networked: server-side database
  - objects
    - extensible, easy to add new fields
  - events
    - check for legal & appropriate 'events'
    - events are undoable user actions
    - AI uses sequence of events (as well as objects) to form design wizard opinions
      - fanciness is up to the implementor

- New since requirements

  - Automated Layout
    - graph representation of objects
    - take into account object tags in laying out graph
    - fanciness is up to the implementor

  - Template Library
    - analogy to word processing programs
      - letter
      - resume, etc.
    - component template (for groups)
      - frontend, backend, GUI, graphics, networking, AI, etc.
    - design pattern tempates
      - command, proxy, state, broadcaster/subscriber, singleton, etc.

- Networking no longer in base spec
  - more focus on layout, AI & code generation
  - but if project isn't large enough:

- Networking specs:
  - all information on a server
  - flow:
    - login/password screen
    - authentication/kerberos
    - load design (search within groups to which you belong)
    - sysadmin version (special password)
      - » assign users to groups
  - post messages
  - timeline
  - checklist
    - ability to "check off" a class from a design to reflect that class is fully "implemented" & ready to use by team members
    - provides a visual representation of team progress
  - member schedules

- Why is networking no longer part of the base spec?
  - based on my research, there aren't many design tools geared toward beginners (& up) out there. However, there are half a million scheduling-timeline-message board-"groupware" products out there that do exactly what the scheduling-timeline-message board-"groupware" parts of my original requirements would do. The standalone version of helloUML, if built tight enough, would be a nice tie-in to an already existing groupware tool.

- as for the cs department handin facility, I feel that it's good enough to simply save a .hml file & use a regular, trusty handin script to submit it to TAs.

- System model in a nutshell (networked)

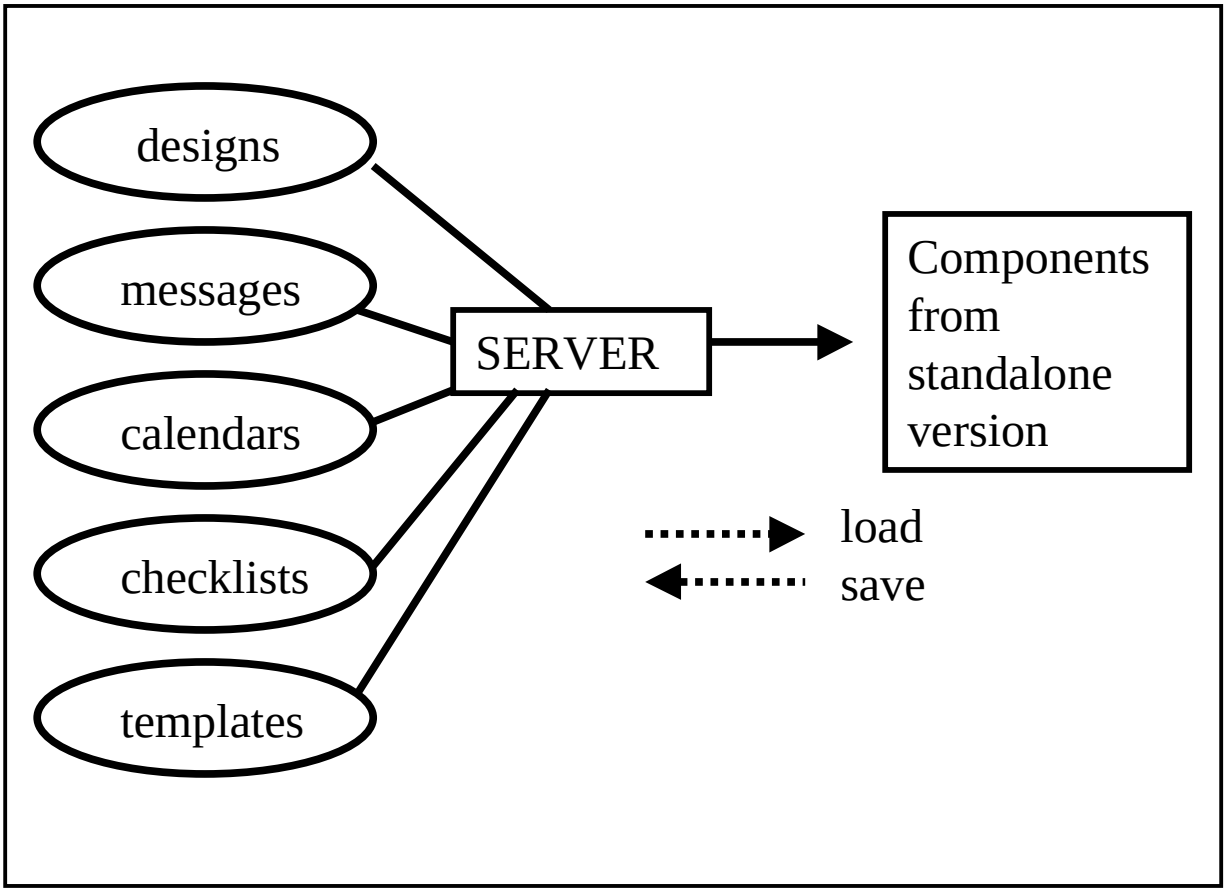


Fig 11. Lame diagram of helloUML's "extended" functionality.

Non-functional Requirements

- testing...1...2...3...
  - random-sample or volunteer user tests (external)
    - if we must test this semester (unlikely since we'll still be coding!), give helloUML to an 8-12 person core group of cs32 students who will work closely with the programming team, offering feedback and sharing gripes whenever possible
    - ideally, give helloUML to an 8-12 person core group of cs15 students to use before the product is released/ recommended to the entire department
    - get sleepy, stressed-out volunteers to try to break the program
    - key assessment to get from volunteer users:
      - is the interface intuitive?
        - » does it make you want to scream? Cry? Yearn for pencil & paper?
- performance
  - each operation should be performed in a manner of seconds
  - navigation/display of UML playground should not significantly slow down when the user's design involves many objects!
    - example test: put 15-20 objects out to play & then start moving the scrollbar. See how long it takes for the screen to refresh.
    - perhaps program can suggest to the user to spread his/her design out over multiple pages for ease of scrollability if #of objects exceeds 20.

- reliability
  - worst case scenario should be: program crashes & work is saved to a crash recovery file

- miscellaneous
  - this program will be built on Sun workstations by 7-9 cs190 students in the C++ programming language during the months of March, April & May, 2000.
  - implementation of this program will not interfere with any functions related to being a "second semester senior."
  - documentation should consist of a small, perhaps pocket-sized user manual that is easily reproducable (i.e., can be photocopied cheaply) for all students taking cs classes.
  - the standalone version of helloUML will not be portable.

- risky business
  - the layout aspect of this program can get very, very complicated. The level of difficulty involved in implementing this feature will depend on the expertise of the team member responsible for it.
  - I don't know enough about AI to be able to accurately assess the level of difficulty involved in creating the design wizard. Like the layout feature, the sophistication of the design wizard largely depends on the expertise of the project team.

- potential-user feedback
  - people I've discussed this project with:
    - the current CS32 TA staff
    - last semester's CS15 TA staff (the majority are now students in CS32)
    - brown grads turned silicon valley programmers (Matt Chotin & Andrea Tartaro)
    - my CS32 group project partner, Elizabeth Irizarry
    - Andy van Dam
    - some guy who interviewed me at Macromedia