helloUML

System Overview

Brown University
CS 190
March 10, 2000

Specifications

Overview

helloUML is intended as a design tool geared towards a specific audience most apply described as an undergraduate computer science department. Although this application is certainly not limited to the Brown University computer science department, all user studies and requirements are the product of Brown University.

The problem with current design tools is that they are not geared specifically for the needs of undergraduate education. Software packages such as Rational Rose (currently used in CS 32) are fully functional UML packages more suited for experienced developers and teams. Undergraduate students just learning about object-oriented programming often get confused, bogged down, and frustrated using these complex applications.

helloUML is a design tool that can be used by even the most novice programmers. By having novice programmers start using design tools from the very beginning, educators can stress the importance of carefully thought-out design. On the other hand, helloUML should not be so limited in functionality that its use is confined to novices. More specifically, the target audience for helloUML is:

- Beginning programmers grappling with design issues (CS15, 16)
- Novice development teams with team design issues (CS32)
- Experienced, larger development teams (CS190)
- Smaller industrial software solutions teams (i.e. consulting firms)

helloUML will be implemented and tested from March through May of 2000 in a team of eight (8) CS 190 students. The package should be available for incorporation in the Brown undergraduate computer science curriculum by fall semester of 2000.

Functional Overview

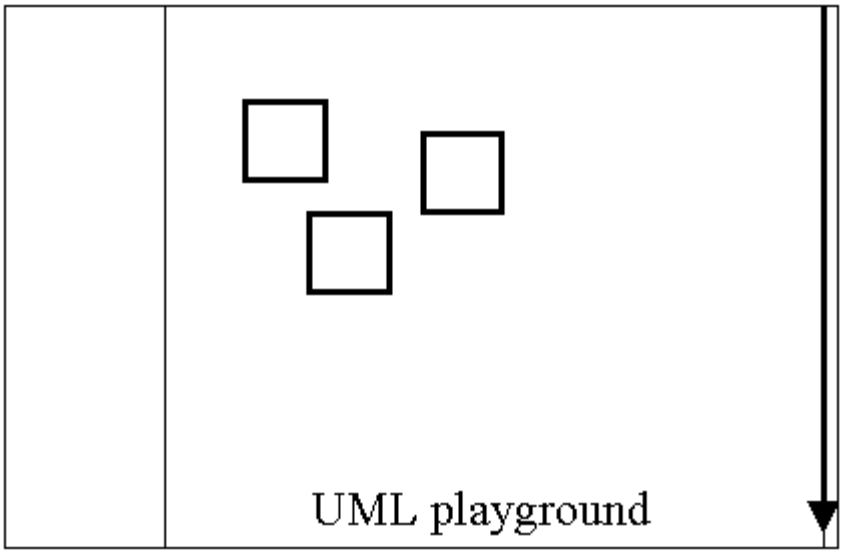
helloUML is a standalone UML diagram tool with the following basic functionality:

- Generate UML diagrams from graphical input
- Logic for object manipulations
- Customizable graphical presentation
- Automated layout for readability and presentation
- Design validity checking
- File and UML generation

User Interface

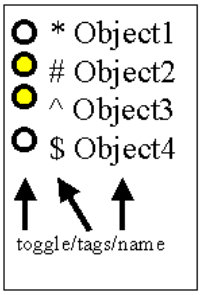
- **Playground**

  This is the main canvas where all objects (classes, text boxes, or other graphical objects) and connections reside. There are scrollbars and graphical manipulation options such as selection, moving, copying, and pasting.



- **Object List**

  Contained on the left side of the application window, this region contains a list of all objects in the playground, a list of tags, and a hide/unhide toggle. This list can be sorted by name, tag, or hierarchy. Selecting an object name in the last selects the object in the playground.

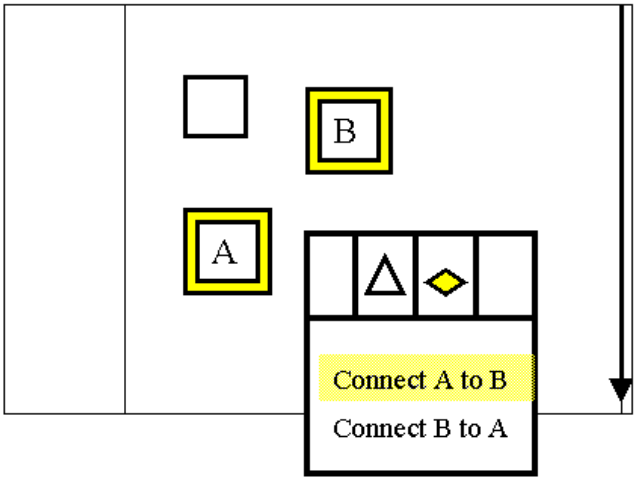


- **Objects**

  There are three basic types of objects – classes, text boxes, and graphical objects (i.e. images). Classes have lists of fields, methods, text strings (i.e. comments), and connection lists (to or from). All of these lists can be toggled on or off. If the user attempts to add a field or method, a dialog will come up for them to enter criteria such as name, return type, protection level, parameters, etc. Any of these fields may be skipped – during code generation, the user will be prompted for missing information. The user can add, remove, or edit any of these entries.

- **Connections**

  Connection are created by clicking on first object A then object B. A dialog will then pop up asking to define the connection.

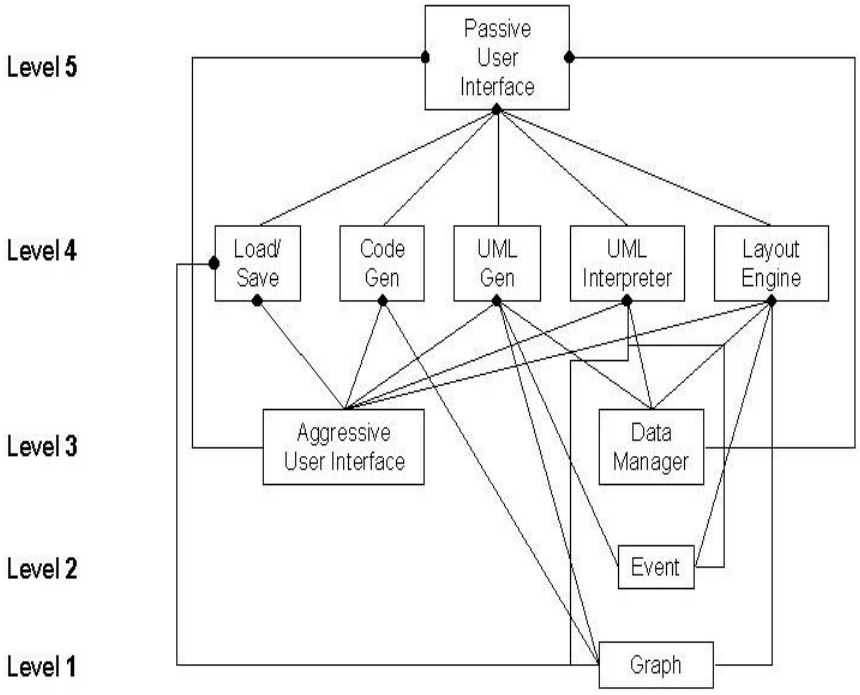


Prioritized Functionality List

Class	Functionality	Priority
CodeGen	Java or C++ - Generate Stubs (i.e. .H files)	1
CodeGen	Parse object methods and variables	1
CodeGen	Generated file doesn't need to compile	1
CodeGen	Check design validity first	4
CodeGen	Allow defaults (i.e. public, void, etc.)	8
File I/O	Open Design	1
File I/O	Save Design	1
File I/O	Open Template	2
File I/O	Save Template	2
Layout	Layout entire diagram	2
Layout	Layout subset (i.e. selected objects)	5
Layout	Take into account tags when laying out	9
Printing	To File	3
Printing	Print Preview	10
Printing	Customizable page breaks & layout	13
UI	Undo	3
UI	Cut/Copy/Paste	7
UI	Customizable GUI - speed buttons and colors	12
UI	Multiple Designs opened simultaneously	16
UI - Object List	Hide/Unhide objects	2
UI - Object List	Select/Focus objects	2
UI - Object List	Manipulate Objects - delete	4
UI - Object List	Customizable sorting & views - alphabetical, by tag, hierarchy	6
UI - Playground	Create connections - click object #1 then object #2	1
UI - Playground	Create/Delete/Modify Objects, fields, and connections	1
UI - Playground	Objects contain methods, variables, and text strings	1
UI - Playground	Scrollbars	2
UI - Playground	Selecting - lasso, tags, object list, all connections to/from	2
UI - Playground	Moving existing objects	2
UI - Playground	Tags w/ customizable colors	2
UI - Playground	Objects - toggle fields, methods, comments and connections	2
UI - Playground	Any drawing object - class, text box, image, etc	4
UI - Playground	Connections connect any object - class, text, or images	4
UI - Playground	Zoom in/out	11
UI - Playground	Create connections - freeform (Floating drawing menu)	15
UML Interpreter	Button to "Check Design Validity"	4
UMLGen	Parse Java or C++ headers - Create Objects	1
UMLGen	Parse Java or C++ headers - Create Connections	3
UMLGen	Fix layout of generated UML	4
UMLGen	Error check file (i.e. files that don't compile)	14

System Model

The following diagram depicts the system modules and their interactions. Detailed descriptions of each component will be explored in the section that follows.



Components

The following section presents a detailed look at each of the system modules, including component interfaces.

Passive User Interface

Consists of menus, floating windows, the object list, the playground, and all event handlers (such as mouse and window refresh handlers). This module handles all graphical interactions between the user and the program and makes appropriate calls to other modules.

Public Methods

- run

Load/Save

This handles all opening and saving of design files as well as templates. Templates are no different than design files, except that class fields, methods, and names are stripped out.

Public Methods

- saveHML
- loadHML
- saveUML
- loadUML
- saveTemplate
- loadTemplate

Code Generation

This module generates C++ or Java interfaces based on the current UML diagram. The UML interpreter will be first called by the passive user interface, so this module can assume that the design is valid and complete. All methods and variables need to be complete, comments included, and connections established. The generated code does not need to compile. For example, if the user defines a function *void Foo(MyType x)* but does not define *MyType* (perhaps it is a typedef), this is acceptable.

Public Methods

- generateCode

UML Generation

The UML generator needs to parse a C++ or Java interface and create a UML diagram. This module need not worry about layout, as the passive user interface will invoke the layout engine. In the initial product version, the code file must be able to be compile (i.e. free of syntax errors).

Public Methods

- generateUML

UML Interpreter

This module is responsible for checking the validity of the design – is this design sufficient to generate code?

Public Methods

- verifyEvent
- verifyGraph

Layout Engine

This module serves as a graph cleanup tool. Layouts can be done on either the entire canvas or subset of the canvas. Layouts may also take into account additional object information, such as tags, classes, or number of connections.

Public Methods

- computeLayout

Aggressive User Interface

This class is a basic dialog manager to be used by all of the inner-level modules. By making this a separate module, UI code can be consolidated, while prevent cyclic dependencies on the passive user interface.

Data Manager

The manager is responsible for handling event transitions and maintaining data integrity.

Public Methods

- getComponentIterator
- processEvent
- undo
- createComponent

Graph

There are two basic types of components – objects and connections. Objects can include classes, text boxes, or graphical objects (images). Connections will use opaque pointers to objects (i.e. the connection will know it links object A with object B). Classes will contain a list of fields, methods, text comments, tags, and connections (to and from). Objects will also contain graphical information such as position, visibility, color, and what fields/methods/connections are displayed. The graph will store a list of connections and a list of objects. Care needs to taken when modifying connection information – the connections contain opaque pointers to their nodes, and the objects contain lists of connections to and from that object. The graph needs to maintain the integrity of this data.

Public Methods

- Accessor functions
- getIterator

Event

This module contains a list of undoable actions, including deleted objects and connections. Events are used to create, delete, or modify data.

Public Methods

- updateComponent
- undo

External Dependencies

The external dependencies of this system are minimal and consist of already implemented and available pieces. These include:

- Graphical Package (yet to be decided)
- Parser/Tokenizer (Lex/Yacc)

Testing

Testing has been broken into three (3) main divisions – module testing, integration testing, and user testing. Technical leads will oversee module testing and the technical implementation of integration testing. Integration testing will be scheduled and coordinated by the project lead (also called the testing coordinator). The testing coordinator will also coordinate user tests.

Module Testing

Each testable class will have a function hu-class_name-Test(). This function will test all testable functions (i.e. functions that are not trivial, accessor functions, basic inline functions, etc.). Within this function, before a function is called, the function name and parameters will be printf'd. After the function returns, the results (i.e. if the function returned the correct value, performed correctly, etc.) will also be printf'd. The exact details of how the function will be tested will be left up to the judgement of the implementers.

Additionally, if the parameters of the function may vary over a range of possible values (such as an integer which may be between 0 and x), the testing function should randomly choose valid values to be passed in, and call the function more than once. Again, the actual implementation will be left up to the implementer.

Stubbed classes may also be needed for this process. Each developer should fill in the stubs with appropriate (i.e. fill in a function in a class that this code depends on to ensure that it calls that function with the proper parameters).

Finally, if a programmer feels that s/he needs debugging code within a function and feels that s/he may need to run this code later, during integration testing, etc., s/he should enclose the code with the compiler directive __HU_-CLASS-_DEBUG__ directive (s/he may have more than one, in which case s/he can number the directives).

Integration Testing

To facilitate integration testing, all component developers should maintain and make available to the team a list of all public functions. This list should contain the function name, parameters, return type, and what exactly the function is suppose to do. In addition, this list should contain all pre- and post-conditions. Finally, the developer should include any additional information deemed necessary (i.e. testing strategies that were employed, known glitches, caution notes, etc.).

We will implement a three stage integration testing process. The process will proceed as follows:

- **Non-functional integration testing** – This happens after the interface is defined and coded. Its main purpose is to make sure that right off the bat, all the pieces fit together as expected. Since none of the functionality is there, we need to stub all the functions, put printf statements in there, and call all the functions from the GUI into the lower modules. This will allow us to assure that the flow of control is correct.
- **Partial-functional integration testing** – Once some of the stubbed functions are be coded and at least a skeletal GUI is up and running, partial integration can begin. In this case, we should be able to bring up the entire application as we would in the end product. The functions that are stubbed should return dummy but correct results, and the functions that are coded should perform what they expect to do. The application as a whole should be functioning, albeit without all the functionality. This stage can actually be performed in parts – instead of testing all the component interactions in one stage, several modules can be isolated.
- **Final integration testing** –.Once base functionality is fully implemented, a final integration test can proceed. In this stage, the developers much systematically test every feature of the application. Do the results seem reasonable? Does the application allow for invalid user inputs? How is performance? The functionality should be frozen at this point. If any additional features are added (possibly due to input from user testing), final integration should begin again to assure that new functionality is fully integrated.

User Testing

Following integration, the testing coordinator will begin to coordinate user tests. A group of approximately ten (10) computer science undergrads will be chosen from varied populations (CS16, 32, and 190). Users will be

asked to use the application to generate mock designs either assigning them predefined tasks (i.e. design a solitaire game) or by having them design a current project they are working on. Users will then asked to be evaluate the software for ease of use, reliability, performance, and usefulness. All participants will be presented with a survey with the following questions:

- Was the interface intuitive? Do you have any suggestions?
- Did you notice any crashes or bugs?
- Did you have any issues over performance?
- Was the documentation useful, not necessary, or insufficient?
- Would you use this tool on your own?
- Do you currently use any other tools that you find better/worse?
- What do you like the most? The least?
- What other features would you like to see?
- Any other suggestion/comments?

Group Organization

Group Roster

Name	Login	Preferred E-Mail	Phone
Joshua Tasman	jmt		272-6719
Michelle Neuringer	mjn		831-0226
Greg Getchell	getch	Greg_Getchell@brown.edu	861-2602
Toan Phan		tpham@netcom.com	272-8245
George Quievryn	gquievry	gq@brown.edu	725-9793
Di Wang		Di_Wang@brown.edu	867-4961
Freddie O'Connell	tfo		831-4626
Chris Filippis		ravenlord@brown.edu	867-4559

Administrative Roles

In accordance with our group culture and overall course goals, every member of the helloUML design and development team will have some role in the group administration and development process.

Name	Login	Title
Filippis, Chris	cjf	Tools
Getchell, Greg	getch	Technical Lead
Neuringer, Michelle	mjn	UI Design, Asst Architect, Research
O'Connell, Freddie	tfo	Chief Architect, Research
Phan, Toan	tphan	Asst Architect
Quievryn, George	gquievry	Project Lead, Testing Coordinator
Tasman, Joshua	jmt	Documention & Communications Manager
Wang, Di	di	Technical Lead

- **Project Lead** – responsible for group organization, setting and enforcing group roles (i.e. making sure everyone knows what they should be doing at all times), setting meeting times, tracking progress, scheduling, and conflict management.
- **Chief Architect** – final top-level design, interface design, approval of component designs, advise project lead on scheduling and progress.
- **Asst Architect** – provide insight and recommendations to chief architect on overall system design.
- **Technical Lead** – establish coding standards, coordinate module testing, communicate progress to project lead, provide technical support.
- **Documentation & Communications Manager** – handle all group communication through website/newsgroup/e-mail postings, maintain

external documentation (reqs, specs, user manuals), maintain internal documentation (interface and component descriptions).

- **Tools** – establish CVS (provide initial training, maintainance, and support), handle makefiles and compilers, provide resources to group members in need of specific tools
- **UI Design** – chose graphical package, design external graphical interface, advise developers.
- **Research** – provide research on specific area needed by project group to be posted to website.
- **Testing Coordinator** – organize integration and user testing

Coding Breakdown

At the onset of project development, module responsibilities will be assigned. The project lead will vigorously track progress to see whether resources need to be reallocated. Any reassignments will be done as early along the development path as is feasible (i.e. as soon as problems are detected).

Module	Developer(s)
User Interface – Passive & Aggressive	Michelle (design), Di, George*
Playground	Michelle (design), Di (principle), George, Freddie*
Load/Save	Greg
Code Generation	Greg
UML Generation	Josh
Layout Engine	Toan
UML Interpreter	Freddie
Data – manager, components, undo	Chris

* - if required

Culture

The following are not meant as rigid laws but merely guidelines. The better we are all able to adhere to the same principles the more smoothly the development process will be.

- **Complaints** – any type of problems, including personal problems, should immediately be brought before the group leader (including complaints about the leader himself).
- **Communication** – Issues between 2 or 3 individuals should be handled through personal communication, either e-mail, phone, or in person. If you have issues that concern the entire group, these should be e-mailed to the project lead. Note – even if you speak with the lead in person or over the phone, send a follow up e-mail to be sure the issue was resolved and the proper people were notified. The lead will take all communications and either act directly or, if the group needs to know, forward this on to the communications manager. Only the communications manager will post to the website. This is important to avoid over accumulation. Avoid sending 100s of one-line e-mails, as these are easier to overlook.
- **Workload** – It is the role of the project lead to be sure that everyone knows their responsibilities at all times. If you are unclear, let the project lead know immediately. In addition, if you find that you have taken on more or less than you think you can handle, communicate this to the project lead immediately. This includes problems with other course load, illness, absences, or over/under estimations. By letting the lead now right away, work can be reallocated before it becomes a problem.
- **Meetings** – meeting times will be arranged in advance so that group members have time to prepare questions, comments, or presentations. The project lead will bring everyone a meeting agenda which will include the meeting goal and itinerary. Members should always try to stick to this agenda. If you have something you want covered at the meeting, e-mail the project lead before so that this can be scheduled in. Any group member is welcome to bring any form of refreshments to any meeting.
- **Progress Reports** – Members are expected to keep a daily record of what they have completed, along with their problems, concerns and questions. These need not be extensive and should not be looked at as a form of personal evaluation. They merely serve as a way of tracking group progress. These should be e-mailed to the project lead each Wednesday. The lead will process these reports and come up with a list of projects that were completed, problems that came up,

solutions that were formulated, questions, goals and deadlines for the following week, and any future meeting times. This will be completed and sent to the documentation manager by Thursday afternoon so that it might be posted to the website. This report will be presented in class that Friday.

- **Decisions** – the project lead (with the help of the entire group) will appoint group decisions (such as design issues, artistic ideas, testing decisions, etc.) to one or more individuals. It is that person/persons job to enlist the suggestions of the group in making this decision. Once all the feelings have been heard, a final decision must be made. Not all people can be made happy all the time – the individual must keep that in mind while trying to arrive at the proper compromise. Everyone must remember that all group members have the group's well-being in mind, and compromises must be made. All final decisions should be honored. If a group member feels the decision to be rash or uncompromising, notify the project lead.
- **Social Activities** – any group member is welcome (and encouraged) to suggest any group social activity. If an event is planned, it should not be viewed as mandatory in any sense – if you are busy, tired, or just plain don't want to attend (for example, you are morally against drinking and the group plans to go to a bar), do not go. You do not need any excuse – just say you won't be able to make it. If you have serious objections to any group activities, let the project lead know. These activities are designed to bring the group together, and no one should be made to feel uncomfortable
- **Absences** – if you can't make any meeting for any reason, let the project lead (or the relevant party) know.
- **Addendums** – if you find any omissions in this document, let the project lead know so that it can be added.

Schedule

- **March 13**
  Solidify system overview – make sure roles, specs, & priorities are clear. CVS should be up and running and group members comfortable with this and any other development tools. The coding standards doc should be completed.

- **March 15**
  Interface proposals, complete with parameters, pre- and post-conditions.

- **March 17**
  Finalize interfaces – start coding. Also need detailed GUI design by this date.

- **April 3**
  Interfaces should be completely stubbed out by this date so that non-function integration can begin.

- **April 5**
  Component designs should be completed by this date and submitted to the chief architect for approval. Architect needs to meet with members individually to discuss and approve designs.

- **April 17**
  Partial functional integration begins. Priorities 1-3 must be complete, particularly a skeletal GUI.

- **April 26**
  Functionality freeze. User tests begin as well as final integration testing.

- **April 28**
  Documentation should be complete.

- **May 3**
  Product completion date. Demo preparation.