CEng 445 Fall 2018 Project Details V2

Contents

Project Details

The following class descriptions may not be final or not the best way to implement the projects. You can modify the following class descriptions if you have a good justification.

Some projects are inherently persistent. Some has weaker relation with unique names. For first two phases, you can assume you know all unique names for object. In later phases you need to implement some basic flat directory based listing of objects so user can choose the object to work with. Also some basic authentication and user ownership can be added in last two phases optionally.

Question Bank for Test Exams in Latex

Question class

`Question` class have the following methods

Method	Description
`constructor (latexbody, choices, topics, parent, embeds)`	create a new question
`getId()`	get unique id for the question
`updateBody(latextext)`	update question body
`addTopic(text)`	add a topic text
`delTopic(text)`	delete a topic
`addEmbed(docid)`	add a document (image or text) embedding
`delEmbed(docid)`	delete a document embedding
`updateChoice(id, latextext, correct, pos)`	update a choice text, if it is a correct choice, positional hint (START or END) of the answer
`updateParent(qid)`	update the parent question, the question that current question is derived from
`updateAskDate(date)`	update the timestamp this question took part in a composed exam, initially `None`
`save()`	Save/update question on database
`copy()`	Create a copy of the question, a new question object is returned
`getLatex(shuffled=False)`	Generate Latex version of the question (without preamble). If shuffled, answers will be in shuffled form, returns lists of correct answer positions

QBank class

Method	Description
`getQuestion(id)`	Get question object by Id
`searchByTopic(topic)`	search question bank for topics matching the parameter
`searchByAskDate(start,end)`	search question bank for questions last time asked in interval, missing `start` means earlier than `end`, missing `end` means later than `start`, both missing means not asked.
`getLatex(iterator, shuffled=False)`	Get an iterator over question ids and generate a Latex exam containing the questions. If shuffled, questions and answers are shuffled. Returns list of correct answers list.
`updateAskDate(iterator,date)`	Get an iterator over question ids and update their last ask date to given date. If date is missing, set time of the call

Other Issues

Each answer choice has a boolean flag denoting the choice is correct, and a positional hint, START or END. This way choices like "all of the above", "none of the below" can be realized. You do not need to implement shuffling at the first phase.

Latex requires an external tool like `pdflatex` to be called. All embedded objects (like pdf, jpeg figures) has to be on the same directory. Also Latex has a preamble commands like used packages etc. like the following:

```
\documentclass{article}
\usepackage{graphicx}
\begin{document}
\input{generatedexam} %%%% This reads generatedexam.tex
\end{document}
```

A sample exam will be given later, in couple of days.

A Simple Logic Circuit Design and Emulator

Component Interface

After a generic logical circuit component interface is defined, any component like gates switches and leds can be define by implementing functions in the interface.

Method	Description
`getName()`	return a string denoting name of the component (like `NAND`)
`getInputs()`	return a list of strings denoting the name of input pins (i.e. `("i0","i1")` for a gate with two inputs)
`getOutputs()`	return a list of strings denoting the name of outputs (i.e. `("out","carry")`)
`getPicture()`	SVG string for drawing the circuit
`calculate(inputlist)`	returns the set of boolean outputs for given set of boolean outputs. For a NAND gate `calculate((True,True))` will return `(False)`

Circuit Class

Circuit object contains all components take part in the circuit and their connections like a graph. Circuit object also keeps track of component id to component mappings.

Method	Description
`constructor(name)`	create an empty circuit with given name or load it from database. Name is a unique identifier for the circuit, given by the user. If `name` exists on database, it is loaded into a circuit object, otherwise an empty circuit with given name is created in memory
`save(name)`	save the name on persistent storage (i.e. sqlite3). If `name` parameter is missing, current name is used
`addComponent(object)`	add a component object implementing `Component` interface in the circuit. Initially it has no connections. Returns a unique id for the object.
`getComponent(cid)`	return the component with given id
`delComponent(cid)`	deletes the component with the given id from the circuit, deleting all existing connections
`connect(cidA, outid, cidB, inid)`	connect the output with `outid` of component `cidA` as input `inid` of the component `cidB`. Output and input ids are the order of the pin in the `Component` interface definition
`disconnect(cidA, outid, cidB, inid)`	break the link that is created by `connect()` as above
`freeinpins()`	Return list of not connected input pins in the circuit as (component id, inid) pairs
`freeoutpins()`	Return list of not connected output pins in the circuit as (component id, inid) pairs
`setinput(inputlist)`	get truth values for all free input pins in the circuit and update whole circuit state, initally all inputs are assumed to be `False`
`getoutput()`	returns a list of truth values for the free output pins

Other issues

You need to implement only binary versions of `AND`, `OR`, `XOR`, `NOT`, `NOR`, `NAND` and `EQUIV` gates as components in the first phase. Only binary versions are required. In the following phase, switch and leds will be implemented as standart components but listed as `freeinpins()` and `freeoutpins()` respectively. SVG component pictures are only required after the third phase.

Your graph will be from output pins to input pins, you can implement any appropriate inner representation. Implementation of `setinput()` requires traversal of the graph. You can make it simple and implement bread-first traversal and update pin values. As long as pin value changes, repeat traversal. Put an upper bound like number of gates over traversal iterations to avoid the infinite loop.

Collaborative XML Docbook

Only a limited subset of Mallard format should be implemented. The elements are:

tag(attr)	Description
page (id, type, xmlns)	can contain info?, title, section*

tag(attr)	Description
section (id)	can contain info?, title, section*, block text
info (id)	can contain desc?, credit *, license, revision
credit(id,type)	can contain name, email*, years?
title (id,type)	inline text
license(id)	inline text
desc(id)	block text
revision	inline text
years	inline text
email	inline text
name	inline text
code(id)	inline text
example	block text +
media(type, mime, src, height, width)	block text
p	inline text
quote	title?, cite? and block text +
comment	title?, cite? and block text +
note	block text+
synopsis	block text+
list(type)	item*
steps	item*
terms	item*
item	block text+
cite(date,href)	inline text
em	inline text
link(xref, hre)	inline text
gui	inline text

Top element is `page`. Inline text does not contain any XML element. Block text can be one of: code, comment, example, figure, list, media, note,p, quote steps terms, synopsis, gui

`?` denotes element is optional + one or more, * 0 or more. A sample document:

```
<page xmlns="http://projectmallard.org/1.0/"
      type="guide"
      id="project-video">
  <info>
    <link type="guide" xref="index#new-project"/>
    <title type="sort">3</title>
    <desc>Write a video to a DVD or SVCD.</desc>
    <credit type="author">
      <name>Ekaterina Gerasimova</name>
      <email>kittykat3756@googlemail.com</email>
    </credit>
    <license>
      <p>Creative Commons Share Alike 3.0</p>
    </license>
  </info>
```

```
  <title>Create a video project</title>

  <p><app>Brasero</app> can be used to create video discs for playing in a DVD
  player or laptop.</p>

  <steps>
    <item>
      <p>Click <gui>Video project</gui> on the start page, or select
      <guiseq><gui>Project</gui><gui>New Project</gui><gui>New Video
      Project</gui></guiseq>.</p>
    </item>
    <item>
      <p>Add the videos to the project by clicking <gui>Add</gui> in the
      toolbar and selecting the files. You can also add files by
      dragging and dropping them onto the project area or by clicking
      <guiseq><gui>Edit</gui><gui>Add Files</gui></guiseq>.</p>
    </item>
    <item>
      <p>You can add a title for the disc in the text entry field below the
      project area.</p>
    </item>
    <item>
      <p>Select the blank disc in the drop down list.</p>
    </item>
    <item>
      <p>Click <gui>Burn...</gui> to continue.</p>
    </item>
    <item>
      <p>Select the <gui>Burning speed</gui> from the drop down list, and any
      other options you may want.</p>
    </item>
    <item>
      <p>Click <gui>Burn</gui> to burn a single disc of the project or
      <gui>Burn Several Copies</gui> to burn your project to multiple discs.</p>
    </item>
  </steps>

</page>
```

You do not have to force constrains for the validity of the elements (section can be inside page, page cannot be inside anything etc) in first phase. You can add it in the following phases.

`xml.etree.ElementTree` is a good class library for parsing and manipulating XML content. It has `XPath` support that you can query the XML tree. For example `"/page/section[1]/media[id='screenshot']"` will select an element with `media` tag under second `section` of the page which has `id` attribute is set as `screenshot`. Also you can use `lxml` which is a seperate library and has extra features.

DBDoc class

`DBDoc` class have the following methods

Method	Description
`constructor`	create a XML document with `page` being the top element
`getId()`	each XML document has a unique id automatically assigned, return it
`setName(name)`	set name of the document. It is `Unnamed` initially.
`getElementbyId(id = '/')`	returns the XML content of the element given by id. If id is `/` whole document XML is returned
`getElementbyPath(xpath)`	returns the XML content of the element given by XPath.
`deleteElement(id)`	delete element with given id and all its children from document
`setElementAttr(id, attr, value)`	set attribute of an element
`setElementText(id, attr, value)`	set inner text (following the tag) of the element
`deleteElementAttr(id, attr)`	delete attribute of an element
`insertChild(id, tag, append = True)`	insert a new, empty element with `tag` as child of the element with `id`. if `append` is `True` it is inserted as the last child, otherwise it is inserted as the first child. It returns id of the new element. Library assigns a unique id for all elements automatically. user can change it with `setElementAttr(oldid, "id", newid)`. If `tag` is `None` there is no XML element but text is to be inserted.
`insertSibling(id, tag, after = True)`	insert a new, empty element with `tag` as immediate sibling of the element with `id`. if `after` is `True` it is inserted as the next element (next child of the parent), otherwise it is inserted as the previous element. It returns id of the new element.
`insertText(id, text)`	add text as the `tail` of the element. Text is placed after the element is closed in the parent.

DBDController class

`DBDController` class provides an interface to `DBDoc` class. There can be multiple `DBDController` objects (ie. clients) accessing the same document. All documents created remain resident until program terminates.

Method	Description
`constructor(id = 'NEW')`	attaches to a `DBDoc`. if id is 'NEW' a brand new `DBDoc` object is created and it is attached.
`upload(xmlcontent)`	it creates a new `DBDoc` object by parsing the content in the parameter

All method calls in `DBDoc` are valid for `DBDController` as well. In phase 2 and later, `DBDoc` needs to keep concurrency in mind.

DBDPersistency class

`SSPersistency` makes sure the SpreadSheets can be saved and loaded on secondary storage. It is a singleton class.

Method	Description
`save(id)`	Saves the current state of document on secondary storage.
`load(id)`	Load the document with given id from secondary storage. If
`there exists a document with same id, it is overwritten.`	
`list()`	list all documents, as id and name tuples on secondary storage
`listmem(dirty = False)`	list all SpreadSheets in memory. If parameter is `True` only SpreadSheets that are changed but not stored are listed.
`delete(id)`	delete SpreadSheet from secondary storage

Other issues

DBDoc and DBDController are observed classes. The calling library can register/unregister callbacks and they are called when DBDoc is updated as in Observer design pattern. In second phase and later, you can use `threading` or `multiprocessing` module notification methods instead of a callback.

An Image Annotation Tool for Shape Based Access Control

LabeledImage class

`LabeledImage` class will store the image and its access labels in a single object. Access control is defined per user or group. Also regular expression based match can be defined. The requesting username is checked against the matches (group, user, or regexp based) in the access control list. Access control list consist of a triple: match expresion, shape and action. If there is a user match, the action is taken as the decision for pixels inside the region. The action can be one of `ALLOW`, `DENY`, or `BLUR`. There is also a default action which is `ALLOW` by default. If there is no match, default action is taken. DENY is realized as black color instead of original image pixel. BLUR is the blurred pixel (take average of neighboring 7x7 subimage assuming current pixel is at the center of the subimage).

In other words for each pixel you need to go over list of rules, if a rule matches by user and shape, take the action. If no rule matches for the pixel, take the default action. Of course, it is better to use black and white images, draw shapes in the given order and use color masking. In order to implement `BLUR` you need a blured version of the image and a blur mask as well.

Method	Description
`constructor()`	Creates and empty image with empty access control
`setImage(buffer)`	Set image content of from a binary buffer. Image can be JPEG of PNG

Method	Description
`loadImage(filepath)`	Set iamge content from a filepath
`load(name)`	Labeled images are stored on a persistent database with a unique name. `load()` overwrites existing data and loads object from persistent storage
`save(name)`	Save current labeled image to persistent storage
`setDefault(action)`	Set default action. Default is `ALLOW`
`addRule(matchexpr, shape, action, pos = -1)`	add an access control rule. `matchexpr` and `shape` is described below. `pos` has default value -1 which means the end of the rule list
`delRule(pos)`	delete the rule in the given position
`getImage(user)`	returns the image assuming the `user` is accessing the image

`matchexpr` is one of `u:username`, `g:groupname`, or `r:regexp`. First version matches the exact username, second version tests if accessing user is in the given group. Third version checks if accessing user name matches the regular expression.

`shape` can be one of `(RECTANGLE, x1,y1, x2,y2)`, `(POLYLINE, [(x1,y1),...,(xn,yn)])`, `(CIRCLE, x,y, r)`.

UserGroup class

This class defines simple user/group relations. A user can belong to mutiple groups and groups consist of multiple users. Users and groups are stored in persistent storage. This class is just object interface to them. `LabeledImage` class refers to user and group names in this class but they are loosly coupled. In case the username in a newly inserted rule does not exist in this class, error is silentyle ignored. Also deleting users/groups does not affect existing labeled images.

Method	Description
`addUser(name, groups, password)`	Add a new user with name and list of groups
`addGroup(name)`	Add a new group with given name
`delUser(name)`	Delete the user with given name
`delGroup(name)`	Delete the group with given name. Existing members of groups should be removed from the group as well
`getGroups(name)`	Get list of groups username belongs to
`getUsers(name)`	Get list of users who are members of the group
`setPassword(user,password)`	Set password of a user
`ismember(user,group)`	Return if `user` is member of `group`

Other issues

You do not have to implement `BLUR` until the third phase. It needs mutiple masks and images.

An Enhanced Log Watch Tool with Custom Filtering

Log monitoring is defind by a group of `LogWatch` objects. `LogWatch` is defined by a resource, source of watch as a file and a set of matching filters. Users insert, delete, modify `LogWatch` objects. Then they activate a set of `LogWatch` object to monitor the logs they are interested in.

As a result there are two operations on this tool: editing and monitoring. At the final phase, there will be multiple users possibly sharing same `LogWatch` objects in their monitoring sessions.

In the first phase, the text files will be used as log resource. In the following phases also socket source will be added.

The filtering rules have a tree structure of your choice since `AND` and `OR` operators can be used to combine match expressions. In order to edit this tree, you need to address an arbitrary node or leaf. A path based addressing can be in form of list of binary branch choices as `(0,1,1,0)` denoting left, right, right, and left branch is traversed to get a subtree. The null tuple `()` denotes the root tree.

LogWatch class

Method	Description
`constructor(filename)`	creates a new logwatch object monitoring the given log file
`setMatch(match, address = ())`	replace the addressed node in the tree with the given `match`
`combineMatch(match, connector, address)`	replace the the addressed node in the tree with a new node with given `connector` (`AND` or `OR`). Left branch of connector will be the previous node value, right branch will be the new `match`.
`delMatch(address)`	delete the node in the given address. The sibling of the node will replace the parent logical operator. deleting the root will clear all filters, matching everything
`register(object)`	register the given object as the list of observing object. In case of a new log with the positive match, the `newlog(logbody)` method will be called on all registered objects.
`unregister(object)`	unregister the previously registered object as the observer
`save(name)`	Save current logwatch with the given unique name to persistent storage
`load(name)`	Load logwatch with the given unique name from persistent storage

The `match` value is a n-tuple in the form `(matchfield, operator, value, negated, caseinsens)`. `matchfield` is one of `WHOLE`, `IP`, `SEVERITY`, `FACILITY`, `FIELD:range:sep`, `RE:regexp:field`. The `FIELD` is followed by a range description, either a single number two numbers seperated by a dash. the last subfield is the separator symbol. For example `FIELD:2-5:`, will seperate log line with `,`, compose a substring from field 2 (starting from 0) to 5 inclusive and match that string. For example `FIELD:3:` (last component is a space) gets the forth word of a log.

`RE:regexp:field` passes syslog message on a regular expression substitute and subtitutes it with the given `field`. This way message body extracts the regular expression based group. The value is simply:

```
re.sub(message, regexp, '\g<' + field + '>')
```

`IP` is ip number or hostname specified in the log. `WHOLE` matches the whole normalized syslog message.

`operator` is one of `EQ`, `LT`, `LE`, `GT`, `GE`, or `RE`. `RE` is used for regular expression match assuming `vaue` is a regular expression. All the others are comparison operators. For `IP` comparison is based on the octets of the IP number, first one being most significant. For `SEVERITY`, `emergency` is the greatest, `debug` is the smallest severity (See manual pages of syslog).

`value` is the other operand of the operator. First operand is the log component.

If `negated` is `True` the calculated match value is reversed.

If `caseinsens` is true all matches are case insensitive, values are converted to lowercase and than compared.

Other issues

In first phase you need to watch a simple log file like `/var/log/syslog` which does not have record structure and misses severity, facility values. IP usually follows the time but others are missing. In the second phase you need to add network (UDP) based logging as a log server. In this case, other computers in the network will send you structured data over network with RFC 5424. Fortunately there are RFC 5424 python parsers available. You can use them.

You need to implement `FIELD:range:sep` and `RE:regexp:field` value formats in after first phase as well.