CEng 445 Fall 2017 Project Details

Contents

1 Collaborative Spreadsheet

Cell addressing is as usual A..Z, AA..ZZ for columns followed by an integer id as A1, JA101. Rectangular ranges can be retrieved as top left cell and bottom right cell separated with column ':'. Number of columns/rows can be fixed in spreadsheet creation. A cell contains either content or a formula.

1.1 SpreadSheet class

`SpreadSheet` class have the following methods

Method	Description
`constructor (rows, cols)`	create a spreadsheet with
`getId()`	each SpreadSheet has a unique id automatically assigned, return it

Method	Description
`setName(string)`	update the name of the spreadsheet. Initially it is `Unnamed`.
`getCell(addr)`	return a tuple (value, celltype). Celltype is one of `'formula'`,`'string'`,`'number'`. For formula type, return a triple. Last element is the formula string
`getCells(rangeaddr = 'ALL')`	return CSV content of the cell range (only values)
`setCellValue( rangeaddr , content)`	set cell content
`setCellFormula( celladdr , formulastr)`	set cell formula
`evaluate(iterations=10)`	evaluate all formula in the SpreadSheet until no significant changes or number of iterations is achieved (in order to deal with cycles)

1.2 SSController class

`SSController` class provides an interface to `SpreadSheet` class. There can be multiple `SSController` objects (ie. clients) accessing the same SpreadSheet. All spreadsheets created remain resident until program terminates.

Method	Description
`constructor(id = 'NEW', rows = 0, cols = 0)`	attaches to an existing SpreadSheet. If id is 'NEW' and rows and cols are non-zero a brand new `SpreadSheet` object is created and it is attached.
`upload(csvcontent)`	it creates a new `SpreadSheet` object by parsing the CSV content in the parameter and attaches to it.
`cutRange(rangeaddr)`	cut range in buffer
`copyRange(rangeaddr)`	copy range in buffer
`pasteRange(topleftcelladdr)`	past values in range on new address (formulas are not required to be modified during paste) Note that CVS format does not have value/formula distinction. In the first phase you only import values. In the second phase and later, import values starting with a '=' as formula.

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

1.3 SSPersistency 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 spreadsheet on secondary storage.
`load(id)`	Load the SpreadSheet with given id from secondary storage. If there exists a SpreadSheet with same id, it is overwritten.
`list()`	list all SpreadSheets ids and names in secondary storage
`listmem(dirty = False)`	list all spreadsheets in memory. If parameter is `True` only SpreadSheets changed but not stored are listed.
`delete(id)`	delete SpreadSheet from secondary storage and memory

1.4 Other issues

SpreadSheet and SSController are observed classes. The calling library can register/unregister callbacks and they are called when SpreadSheet 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.

You can use an equation parsing library like `Equation` for parsing formulas. Standard functions like `abs, floor, ceiling, sin, cos, tan, pow, log, if` are sufficient in a formula. As aggregate functions, `sum, average, countif` are sufficient.

2 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*
`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

tag(attr)	Description
`link(xref, href)`	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>
  <section>
  <title>Preamble</title>
  <p> This is preamble </p>
  </section>
  <section>
  <title>Steps of project</title>
  <steps>
    <item>
      <p>Click <gui>Video project</gui> on the start page, or select
      <gui>Project</gui><gui>New Project</gui><gui>New Video
      Project</gui>.</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
      <gui>Edit</gui><gui>Add Files</gui>.</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>
  </section>

</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`.

2.1 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
`getElementAttr (id, attr)`	get attribute of an element
`getElementText (id)`	set inner text (following the tag) of the element
`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.

2.2 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.

2.3 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

2.4 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.

3 Academic Paper Library

The following information needs to be defined for a paper. In addition to its content:

- Type (one of article, book, incollection, inbook, inproceedings, manual, mastersthesis, phdthesis, proceedings, misc, techreport, unpublished)
- List of authors
- Title of the paper
- Title of the book/collection containing the paper
- Journal name for the paper
- Chapter/Volume/Number for the paper
- Pages of the paper
- Publisher

- Address (City, Country, Institution)
- Date
- Notes, extra information, reminders
- Reference (URL to download or DOI for paper)

3.1 Paper Class

`Paper` class is used to provide an object interface to storage and has the following methods:

Method	Description
`constructor()`	creates a new paper. Assigns a unique id.
`upload(content)`	uploads the paper to the system. The inverted document index is created/updated for content based search.
`getid()`	get paper id
`getmeta()`	get bibliography information as a dictionary
`getcontent()`	download paper.
`setmeta(dict)`	set bibliography information from the dictionary

3.2 PControler Class

`PController` class is used to access papers in the application.

Method	Description
`getPaper(id)`	get paper with the given id.
`searchbyTitle(str)`	search and return list of paper id/author/title tuples for title containing the string. Searches are case insensitive.
`searchbyAuthor(str)`	search and return list of paper id/author/title tuples for authors containing the string.
`searchbyContent(str)`	search papers for content in inverted document index
`searchScholar(author='',title='', num = 10)`	search in Google Scholar for author and/or titles. Return `num` matches with their bibliography information as dictionary objects. User could select one of the entries to call `setmeta()` for easily enter paper data.
`deletePaper(id)`	delete a paper.
`watchPaper(type="NEW|ID|KEYWORD", str, callback)`	watch the database for paper update events by other users. 'NEW' is whenever a new paper is uploaded, 'ID' whenever a paper with given id is updated, 'KEYWORD' whenever a paper with given keyword is uploaded. (You need to implement only one of three types in phase 1, leave others to phase 2)

3.3 Other issues.

You can use `scholar.py` for Google scholar search and getting bibliography information. You can use `textract` to get text out of a document.

For inverted index, you need to keep a mapping for each word to current paper id on a database. A word will appear more than one document, so given word, a list of paper ids will be returned. In a multiword search, result will be intersection of all results. When document is deleted, the related mappings should be deleted as well.

4 Traffic Simulation

Traffic simulation application has two phases. In editor phase, the map and traffic environment is edited. In simulation phase, cars arrival parameters are set and simulation started.

4.1 Map Class

Map class contains roads and junctions in two dimensional world. Basically it is a graph with edges have properties about the road capacity and edge weight is the euclidean distance between the vertices in 2D. `x` dimension in graph is horizontal, west to east increasing positive floating point values. `y` dimension is vertical south to north increasing positive floating point values. In other words bottom left corner of the map is the origin.

Roads/edges have number of lanes as an attribute which is an integer from 1 to 3. The edges are directed. Vertices are addressed by a user specified id where edges are addressed as a tuple, `((start, end)`

Method	Description
`constructor()`	create an empty map
`addNode(id, x, y)`	create a vertice at (x,y) with given id.
`deleteNode(id)`	delete the vertice with given id. All edges connecting the vertice are also deleted
`addRoad(id1, id2, nlanes = 1, bidir = True)`	add an edge from id1 to id2. Number of lanes is set for each edge and if `bidir` is `True` another edge in opposite direction is also created.
`deleteRoad(id1, id2, bidir = True)`	delete the edge between id1 and id2. If `bidir` is `True`, the edge in opposite direction is also deleted.
`getShortestPath(id1, id2)`	The list of edges in the shortest path from id1 and id2 is returned as a list of tuples. (You can use Floyd-Warshall algorithm to calculate all shortest paths when this is called for the first time, then use cached value as long as Map is not changed)
`saveMap(name)`	Save map with given name from database
`deleteMap(name)`	Delete map with given name from database
`loadMap(name)`	Load map stored in database, replace current map

4.2 Simulation Class

For generating artificial traffic, simulation has a list of traffic generators. Each generator will produce a random traffic once in the given period. The traffic is defined by a list of source nodes and a list of destination nodes. Generator picks a random node from first list, and another for second. Then gets the shortest path from map and inserts a vehicle on the path. Generators can be easily implemented as threads in a loop as:

```
iterate number times:
    wait for period ticks
    pick random node1 from list1
    pick random node2 from list2
    path = Map.shortestpath(node1, node2)
    insert path as a vehicle on first road segment
```

Method	Description
`constructor()`	Creates an empty simulation
`setMap(map)`	set Map object as the map for the simulation
`addGenerator(sourcelist, targetlist, period, number)`	Add a traffic generator. Generator generates a vehicle once in the given period. After generating number vehicles it stops generating.
`getGenerators()`	get a list of generators and their parameters
`delGenerator(num)`	delete the generator in the given order of insertion (`getGenerators()` list order)
`startSimulation(tickperiod = 0)`	Start the simulation with each tick of clock lasts `tickperiod` milliseconds. A thread sends tick() signal to all generators and edges to advance simulation once in `tickperiod` milliseconds. If `tickperiod` is 0, tick() is explicit. The program/user calls `Simulation.tick()` that advances simulation clock explicitly.
`tick()`	explicit advance of simulation. `tick()` signal is generated (methods are called) for each generator and simulation component.
`terminate()`	end the simulation
`wait()`	wait for the end of simulation. If manual `tick` it returns if simulation is over.
`getStats()`	get simulation statistics. Total number of vehicles, number of vehicles completed, number of vehicles currently on map, average speed, maximum vehicles per road segment, current vehicles per road segment, average speed per segment. Road segment statistics need not to be complete in phase 1.

4.3 RSegment Class

Implements behavior of a road segment connecting two nodes/vertices during simulation. Each road segment works as a queue of vehicles. When a vehicle complete a segment, it is inserted in the new segment in its path and deleted from the current segment. When a vehicle is inserted in a segment it has an expected completion time for that segment. This time depends on the length of the road and speed of the car. The speed depends on how crowded road segment is.

speed = fcrowd(nvehicles, nlanes, length) * kspeed

```
def  fcrowd(v,l, d):
    '''returns a speed factor based on how crowded road segment is
    it returns fmax for all values in [0,cmin]. fmin for all values > cmax
    all intermediate values are linearly interpolated'''
    fmin,fmax = 0.05, 1.0
    cmin,cmax = 10, 100
    c = v/(l*d)      # vehicle per unit distance in a lane
    if c <= cmin: return fmax
    elif c < cmax: return fmax - (c - cmin) * (fmax-fmin)/(cmax-cmin)
    else: return fmin
```

kspeed is the base speed of a car in an empty road segment, which is 60.0, you can make it a configurable item in Simulation class. nvehicles is the vehicles in front of the current vehicle.

In order to simulate vehicles you have two options:

1. Fix the expected leaving time on entrance as now+speed*length. When this time is reached, make vehicle leave the segment.
2. At each simulation tick, advance the position of the vehicle in the road by its current speed (based on formula above speed will be dynamic based on nvehicles) when vehicles reaches segment length, it leaves the segment.

You can choose any of them. First one is simpler but vehicles from back can go faster and leave the segment before.

A road segment also have a capacity 150 vehicle per unit time per lane. In other words:

segmentcapacity = ceiling(150 * length * nlanes)

When a segment has this many vehicles, next vehicle trying to enter the segment needs to wait for a vehicle to leave the segment. You do not need to implement segment capacity in phase 1. It should be available in phase 2.

Class methods are:

Method	Description
`constructor(~~len,~~ nlanes, edge)`	construct a segment with floating point length and number of lanes for the map edge. `edge` is the edge information as you represent in the Map. length is calculated from map.
`insertVehicle(vehicle)`	insert vehicle. You are free in how you present a vehicle, you can implement a class for it or represent as a dictionary. It needs to keep track of remaining road segments in its path and speed/timing information (start time, completed length, completion time or current position)
`getInfo()`	return length, number of lanes and nodes it connects in a tuple.
`getNVehicles()`	return number of vehicles in the segment
`getCapacity()`	return capacity of the vehicle
`full()`	check if capacity is reached (`getNVehicles() >= getCapacity()`)
`waitCapacity()`	wait for a new opening for capacity (in phase 2)
`getStats()`	return statistics about the segment, max vehicles, average speed (for completed vehicles), current number of vehicles. (for phase 2)

4.4 Other issues.

Assume each Map object is only edited by a single user. Also each Simulation is setup and executed by a single user. Each user connecting the system creates its own isolated instance.

In phase 2 and later, a simulation will be interactive, user may request to get simulation statistics periodically. Statistics are pushed to user automatically, not pulled by the client.

5 Location Based Events

In this project, latitude, longitude based real world coordinates will be used. The latitude is a floating point value between -90 to 90, from south pole to the north pole. The longitude is a floating point value between -180 to 180, west to east. Our department is at latitude 39.8918, longitude 32.7835. In last phases you will be using map services like openstreetmap, leaflet.js etc. to present events on the real world map.

5.1 EventMap Class

EventMap class will implement the container for events. Users can create multiple maps and more than one user can attach to the same map. For example there might be an academic events map and concerts map, each having tens of users connected. A user can attach to only one map.

Method	Description
`constructor()`	Create an empty map without any events.
`insertEvent(event, lat, lon)`	inser event at latitude longtute.
`deleteEvent(id)`	delete event with id
`eventUpdated(id)`	an event calls this function when it is updated to inform the container map.
`searchbyRect(lattl,lontl, latbr, lonbr)`	search events in the rectangle defined by top left and bottom right coordinates.
`findClosest(lat, lon)`	find closest event to the coordinate
`searchbyTime(from, to)`	search events between date+time info specified in parameters. Parameters are strings in exact form '2017/11/03 13:43'. Also strings in the form +num (hours | days | minutes | months) can be given. `'+0'` stands for now, `'+1␣months'` stands for next month today, at current hour, at current minute. Match is based on overlap. If parameter [from–to] and event [from–to] has at least one second in common, it is considered as a successfull match.
`searchbyCategory(catstr)`	search events with catstr is included in events category list.
`searchbyText(catstr)`	search if event text (title, description, or location) contains the parameter string. Search will be case insensitive.
`searchAdvanced(rectangle, from, to, category, text)`	search for all parameters combined. if a parameter is specified as `None`, all events match for that criteria. You can implement other searches in terms of this function.
`watchArea(rectangle, callback, category = None)`	registers a callback for all event modifications in the rectangular area. When a new event is inserted, deleted or event information changed in the rectangle, the callback is called as `callback(type, event)`. The type can be `INSERT, DELETE, MODIFY`. If `category` parameter is not none events are filtered by the specified category.

5.2 Event class

Method	Description
`constructor(lon,lat, locname, title, desc, catlist, from, to, timetoann)`	creates an event with given coordinates, location name, event title, description, category list (string list), start time, end time and time to announce.
`updateEvent(dict)`	The event is updated by the fields in dictionary. Fields are `latitude, longitude, locname, title, description, categories, from, to, timetoann`.
`getEvent()`	returns a dictionary as above to get all information about the event
`setMap(mapobj)`	when an event is inserted in the map, this method is called to associate event with its container map. An event can only belong to one map.
`getMap()`	return the map that event belongs to

An event remains not announced until its time to announce is reached. If time to announce is not specified, it is in announced state by default.

5.3 EMController class

This class is an interface to control an `EventMap` object.

Method	Description
`constructor(id = 'NEW')`	attaches to an `EventMap`. If id is 'NEW' a brand new `EventMap` object is created and it is attached.
`dettach()`	dettaches the `EventMap` object. All watches will be cleaned up.
`save(name)`	Save controlled map and all events in it on secondary storage.
`load(name)`	Load and create a brand new `EventMap` from secondary storage and return id of it. This is a class method (called as `EMController.load(name)`).
`list()`	return a list of all map object names on secondary storage. This is a class method.
`delete(name)`	delete a map object from secondary storage. This is a class method.

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