CSE535 PHASE 2: PSEUDO-CODE FOR ORIGINAL ALGORITHM

Team: Atanu Ghosh - 110280569, Muhammad Ali Ejaz - 110559131

Master

```
// Master is from where everything is started. All the coordinators, database
// and loggers are set up
main()
      // Read from config file
      // Set up co-ordinators
      // Set up clients.
      // Setup loggers.

      // Start DB Instance
      // Start Coordinators
      // Start Clients

      // Wait for all Clients to complete and then
      // send done signal to other processes
```

Client

```
LIST_OF_COORDINATORS = <list of all coordinators>
// Read config.ini and generate list of requests and store in a queue
requestQueue = getSequenceFromConfigFile(configFile)
while !requestQueue.empty()
      request = requestQueue.pop()
      coordinator = hashFunction(request.subjectID, LIST_OF_COORDINATORS)
      send(request, to= coordinator)

receive(response)
      if response.result = True
            // Allow access
      else
            // Deny Access
```

Coordinator

```
// Coordinator class is same for subject as well as resource coordinators
// because a coordinator can act as a subject coordinator for some subjectID
// and a resource coordinator for some other reourceID
```

```
setup()
      // This function is called when the coordinators are started. It starts
      //    workers and initalizes the caches
      tentativeSubjectCache = {}
      mainSubjectCache = {}
      resourceCache = {}
      requestQueue = {}
      responseQueue = {}

// evaluatePolicy (receive) is used by subject coordinator and is the entry
//    point from client instance.
// subjectID and resourceID are the IDs of subject and resource sent by
// Application instance
receive(subjectID, resourceID, action, from= client)
      // set up administration for subject
      requestID = getUniqueID()

      // Add this request to requestQueue
      requestQueue.append(requestID)

      // get resource coordinator and call resource coordinator's authorize
      // method. A request structure or a class will be used to construct
      // the request
      resourceCoordinator = hashFunction(resourcID, LIST_OF_COORDINATORS)

      request = struct RequestStruct {
            subjectID
            resourceID
            evaluationID
            tentativeUpdateSubjectMap
            cacheResourceMap
            subjectAttributeSetRequired
            resourceAttributeSetRequired
      }
      send(request, to= resourceCoordinator)
```

```
// subject coordinator method that receives result from worker
// recieve would be called by the listener, listening for response
// from worker
receive(response, from= worker)
      // If response is not for the first request push to response queue
      //    and return
      if (requestQueue.top().requestID != response.requestID)
            responseQueue.push(response.requestID, response)
            return

      checkAttributeConflicts(response)

// Check for conflicts in maps and current cache and determine
// whether we should update or re-evaluate the request.
// would be called from subject coordinator
// after subject conflict check sends the response to resource coordinator
// for further resource conflict check
bool checkAttributeConflicts(response)
      // equal, subset and intersect would compare key
      // as well as value in map!
      // if tentativeUpdateMap equals currentCacheMap
      //    that is changes were done on same map, no conflict
      // OR
      // if the map changed but there's no conflict between
      //    the cached attributes and used attributes in worker
      //          merge and update current cache
      // else
      //          conflict happened, re-evaluate the request

      if (response.evaluationResult == False)
            send('false', subjectID, resourceID, action),
                                                      to=fromClient)
            if (requestQueue.top().requestID in responseQueue)
                  response = responseQueue[requestID]
                  delete responseQueue[requestID]
                  checkAttributeConflicts(response)

      // readDbSubjectMap contains the subject attributes read from DB
      for each item in readDbSubjectMap
```

```
            if item exixts in subject attribute cache
                  conflictHappened = True

      // readSubjectAttributeMap contains the subject attributes read from
      // the cache sent to worker
      for each item in readSubjectAttributeMap
            if item exists in subject attribute cache and the timestamp for
            the same item in the two maps are not same
                  conflictHappened = True

      if (conflictHappened == True)
            // remove this request from requestQueue
            delete requestQueue.top()
            // restart the reqest
            send(subjectID, resourceID, action, to= subject_coordinator)
            if (requestQueue.top().requestID in responseQueue)
                  response = responseQueue[requestID]
                  delete responseQueue[requestID]
                  checkAttributeConflicts(response)
                  // returns

      // No conflicts Happened
      // toUpdateSubjectMap contains all subject attribute that needs update
      for each item in toUpdateSubjectMap
            tentativeSubjectCache[subjectID][item.key] =
                                                      (item.value, timestamp)

      send(response, to= resource_coordinator)

// Check for conflicts in resource attributes and determine
// whether we should update or re-evaluate the request.
// would be called from subject coordinator
// after resource attribute conflict check sends its result back
// to subject coordinator
recieve(response, from= subject_coordinator)
      // readDBResourceMap contains the subject attributes read from DB
      for each item in readDBResourceMap
            if item exixts in resource attribute cache
                  conflictHappened = True

      // readResourceAttributeMap contains the resource attributes read
      // from the cache sent to worker
      for each item in readResourceAttributeMap
```

```
            if item exists in resource attribute cache and the timestamp for
            the same item in the two maps are not same
                  conflictHappened = True

      if conflictHappened == False
            // toUpdateResourceMap contains all resource attribute that needs
            // update
            for each item in toUpdateResourceMap
                  resourceCache[subjectID][item.key] =
                                                      (item.value, timestamp)
            send(toUpdateResourceMap, to= database)

      send(response, conflictHappened, to= subject_coordinator)


// Receives resource attribute conflict check response from resource
// coordinator.
recieve(response, conflictHappened, from= resource_coordinator)
      if (conflictHappened == True)
            // delete tentativeSubjectCache for this subject
            tentativeSubjectCache[subjectID] = {}
            // remove this request from requestQueue
            delete requestQueue.top()
            // restart the reqest
            send(subjectID, resourceID, action, to= subject_coordinator)
      else
            for each item in tentativeSubjectCache
                  mainSubjectCache.update(tentativeSubjectCache)
            // empty tentativeSubjectCache for this subject
            tentativeSubjectCache[subjectID] = {}
            send('succes', subjectID, resourceID, action, to= client)
            send(toUpdateSubjectMap, to= database)

      // serve next request
      if (requestQueue.top().requestID in responseQueue)
            response = responseQueue[requestID]
            delete responseQueue[requestID]
            checkAttributeConflicts(response)
```

```
// Resource coordinator instance.
// Receives request from subject coordinator
receive(request, from= subject_coordinator)
      // Get a worker (round robin manner) to evaluate the current request
      worker = getWorker()
      // execute request evaluates the actual policy
      send(request, to= worker)
```

Worker instance

```
// Worker does the actual policy evaluation
setup()
      policy = readPolicyFromPolicyFile()

// execute evaluates the policy for the given request
receive(dbresponse, from= database)
      // Some central store which will have XACML policies
            response = struct ResponseStruct {
                  request
                  dbresponse
                  tentativeUpdateSubjectMap
                  tentativeUpdateResourceMap
                  workerUpdateSubjectMap
                  workerUpdateResourceMap
            }
            responseResult = validateRule(request, dbresponse)

      // send response to subject coordinator
      send(responseResult, to= subjectCoordinator)

receive(request, from= coordinator)
      // Populate required subject and resource attributes
      //    into request based on request.action
      send(request, to= database)


response validateRule(request, dbresponse)
      // Goes through all the rules in policy and check if anything matches
      // If anything matches we break and perform any updates for that rule
      rule = struct PolicyRule {
            subjectCondition
            resourceCondition
```

```
        subjectUpdate
        resourceUpdate
}

found = False
for rule in policy[request.action]
        for elem in subjectAttributeSetRequired:
                if rule.subjectCondition == elem:
                        for elemRes in resourceAttributeSetRequired
                                if rule.resourceCondition == elemRes:
                                        found = True
                                        break


if found == False:
        pass
else:
        // 'rule' is our policy rule
        response.workerUpdateSubjectMap =
                updateAttribs(rule.subectUpdate)
        response.workerUpdateResourceMap =
                updateAttribs(rule.resourceUpdate)

return response


updateAttribs(updateMap)
        result = {}
        for updates in updateMap
                // Read rule and perform update
                result[updates.key] = result.value
                // Increment updates.value if ++
                // Decrement updates.value if --
        return result
```

Database Instance

```
// This is in-memory database instance which is read from a dbInit file.
//    It contains 2 maps of subject and resource attributes.

setup(config) // config object after readin config.ini file
        dbInitFile = str(config.get("setup", "dbInitFile"))
        minDbLatency = int(config.get("setup", "minDbLatency"))
        maxDbLatency = int(config.get("setup", "maxDbLatency"))

        // XML file parser
        root = ET.parse(dbInitFile).getroot()
        subjectMap = {}
        resourceMap = {}
        // Initialize a random wait time after which it must commit
        waitTime = random.randint(minDbLatency, maxDbLatency)


// This receive funtion, receives a request from worker fetches the required
attribtes and thir values from DB and returns
receive(request, from= worker)
        subjectID = str(request.subjectID)
        resourceID = str(request.resourceID)

        // If no subjectID in DB then write it with empty values
        if subjectID not in subjectMap
            subjectMap[subjectID] = {}
        if resourceID not in resourceMap
            resourceMap[resourceID] = {}

        subjAttribDiffSet = request.requiredSubjAtrribs - set(
                request.subjectAttributeMap.keys())
        resourceAttribDiffSet = request.requiredResourceAtrribs - set(
                request.resourceAttributeMap.keys())

        for subj in subjAttribDiffSet:
            try:
                  subjAttribDiff[subj] = subjectMap[subjectID][subj]
            except:
                    // If attribute not present write empty value
                    subjectMap[subjectID][subj] = ""
                    subjAttribDiff[subj] = ""

        for resource in resourceAttribDiffSet:
            try:
                 resourceAttribDiff[resource] = resourceMap[resourceID][
                        resource]
            except:
                // If attribute not present write empty value
                resourceMap[resourceID][resource] = ""
                resourceAttribDiff[resource] = ""

        dbresponse = DBResponse(request, evaluation, subjAttribDiff,
                                resourceAttribDiff)
        send(response, to= worker)


// This receives, receives from coordinator and updates the updateMap
// with the exisiting contents of the map (database).
receive(updateMap, from= coordinator)
      // Wait for waittime before commiting to DB
      Timer(waitTime, updateFunc, [updateMap]).start()

updateFunc(updateMap, instance):
        if (updateMap.instance == "resource"):
            resourceMap[updateMap.key].update(updateMap.value)
        else:
            subjectMap[updateMap.key].update(updateMap.value)
```