```
class appRequest {
    string subCoordinator
    string resCoordinator
    string subjectId
    string resourceId
    string evaluationId
    timestamp reqTimeStamp
    subjectAttribute tentativeAttr
    string tentativeEvalId
    enum type
    string result
}

class subjectAttribute {
    timestamp time
    subAttributes attr
    string evaluationId
}

hashMap <evaluationId, request> subEvalList
hashMap <evaluationId, resourceAttribute> resEvalList

hashMap <subjectId/resourceId, coordinatorId> coordinatorMap
hashMap <subjectId, subjectAttribute> subjAttrMap
hashMap <resourceId, subjectAttribute> resourceAttrMap
hashMap <subjectId, subjectAttribute> tentativeMap

//Application side
sendRequest(subjectId, resourceId) {
    appRequest request = allocateNewRequest(subjectId,resourceId)
    //initialize new request
    request.subCoordinator = coordinatorMap[subjectId]
    request.resCoordinator = coordinatorMap[resourceId]
    request.reqTimeStamp = getCurrentTime()
    request.type=POLICY_EVAL_REQUEST
    result = sendPolicyEvalautionRequest(request,request.subCoordinator)
    //send request to corresponding subject co-ordinator
    //handle result
}

checkForConflict(appRequest request, string type) {
    if(type == subject) {
        currAttr = subjAttrMap.get(request.subjectId, request.evaluationId)
        for(item in subjAttrMap) {
            if(attribute in currAttr matches any attribute in item)
            //If any attribute has been accessed by any other request
                return conflict = true
        }
        return conflict = false
    }
    else if(type == resource) {
        currAttr = resourceAttrMap.get(request.resourceId, request.evaluationId)
        for(item in resourceAttrMap) {
            if(currAttr.attr matches any attribute in item.attr)
            //If any attribute has been accessed by any other request
                return conflict = true
        }
        return conflict = false
    }
}

//Subject Coordinator side
handleRequest(appRequest request) {
    if(request.type == POLICY_EVAL_REQUEST) {
        request.evaluationId = generateUniqueEvaluationId()
        subEvalList.add(request.evaluationId, request)

        subjectAttribute tentativeAttr = tentativeMap[request.subjectId]

        //if tentative attributes exist for this subjectId
        if(tentativeAttr) {
            request.tentativeAttr = tentativeAttr.attr
            request.tentativeEvalId = tentativeAttr.evaluationId
        }

        subjectAttribute subjAttr = allocatesubjectAttribute()
        subjAttr.attr = initializeSubjAttr(request)
        subjAttr.tentative = false
        subjAttr.evaluationId = request.evaluationId
        subjAttr.time = request.reqTimeStamp
        subjAttrMap.add(request.subjectId, subjAttr)
        //initalize attributes for this subject and add them to map

        sendPolicyEvaluationRequest(request, request.resCoordinator)
        //send request to corresponding result co-ordinator

    }
    else if(request.type == POLICY_EVAL_RESULT) {
        if(request.result == SUCCESS) {
        //If constraint check from worker success
            if(checkForConflict(request, subject) == false) {
            //If no conflicts between subject attributes
                if(request.tentativeAttr != NULL) {
                //If this request read from tentative data
                    while(tentativeMap[request.subjectId])
                        wait()
```

```
                        //wait till tentative data is committed or discarded
                    if(tentativeMap[request.subjectId] has been committed) {
                    //If tentative data was committed
                        subjectAttribute subjAttr = subjAttrMap.get(request.subjectId, request.evaluationId)
                        tentativeMap.add(request.subjectId, subjAttr)
                        //add current data to tentative
                        request.type=CONFLICT_CHECK
                        sendConflictCheckRequest(request, request.resCoordinator)
                        conflictReqRcList.add(request.resCoordinator)
                    }
                    else {
                    //If tentative data was not committed
                        request = getOriginalRequest(request)
                        //retrieve original request using subjectId and evaluationId
                        sendPolicyEvaluationRequest(request, request.resCoordinator)
                        //resend request
                    }
                }
                else {
                //If request did not read from tentative data
                    subjectAttribute subjAttr = subjAttrMap.get(request.subjectId, request.evaluationId)
                    tentativeMap.add(request.subjectId, subjAttr)
                    //add current data to tentative
                    request.type=CONFLICT_CHECK
                    sendConflictCheckRequest(request, request.resCoordinator)
                    //send conflict check request to corresponding resource coordinator
                    conflictReqRcList.add(request.resCoordinator)
                    //Store resource coordinator id in a order in list
                }
            }
            else {
            //If there were conflicts
                request = getOriginalRequest(request)
                //retrieve original request using subjectId and evaluationId
                sendPolicyEvaluationRequest(request, request.resCoordinator)
                //resend request
            }
        }
        else {
            //If constraint check from worker failure
            sendPolicyEvaluationResultToApp(request.result)
            //send failure to app
        }
    }
    else if(request.type == CONFLICT_CHECK) {
        conflictReplyRcMap.add(request.resCoordinator,request)
        //key - rc Id, value - reply from rc
        if(request.resCoordinator != head(conflictReqRcList)) {
        //check if reply is first request sent to rc. If not return
            return
        }
        if(request.result == SUCCESS) {
        //Commit in the order the tentative update is executed
            do {
                request = conflictReplyRcMap.get(request.resCoordinator)
                commit(tentativeMap[request.subjectId])
                tentativeMap.remove(request.subjectId)
                removeRequestFromSubjectAttribute(request)
                updateHead(conflictReqRcList)
            } while(head(conflictReqRcList)!=NULL && conflictReplyRcMap.get(request) == head(conflictReqRcList) )
        }
        else {
        //restart the evaluation again if rc conflict is positive
            do{
                request = conflictReplyRcMap.get(request.resCoordinator)
                request = getOriginalRequest(request)
                sendPolicyEvaluationRequest(request, request.resCoordinator)

                for(item in subEvalList) {
                //restart the associated dependent evaluation requests also
                    if(item.value.tentativeEvalId == request.evaluationId)
                        sendPolicyEvaluationRequest(request, request.resCoordinator)
                }
            }while(head(conflictReqRcList)!=NULL && conflictReplyRcMap.get(request) == head(conflictReqRcList) )
        }
    }
}

//Resource Coordinator side
class resourceAttribute {
    timestamp time
    resAttributes attr
    string evaluationId
}

handleRequest(appRequest request) {
    if(request.type == POLICY_EVAL_REQUEST) {
        if(resEvalList[request.evaluationId]) {
        //If request is ongoing but received again
            resourceAttribute resAttr = resourceAttrMap.get(request.resourceId, request.evaluationId)
            resAttr.attr = initializeResAttr(request)
            resourceAttrMap.update(request.resourceId, resAttr)
            //restore resource attributes from original request
        }
        else {
            resEvalList.add(request.evaluationId, request)
            //If new request, then initialize resource attributes and add it
```

```
194.
195.                 resourceAttribute resAttr = allocateresourceAttribute()
196.                 resAttr.attr = initializeResAttr(request)
197.                 resAttr.evaluationId = request.evaluationId
198.                 resAttr.time = request.reqTimeStamp
199.
200.                 resourceAttrMap.add(request.resourceId, resAttr)
201.             }
202.             assignToWorker(request)
203.             //Send request to worker for evaluation
204.         }
205.         else if(request.type == CONFLICT_CHECK) {
206.             if(checkForConflict(request, resource) == false) {
207.             //If request's conflict check failed, send success or failure
208.                 request.result = SUCCESS
209.             }
210.             else {
211.                 request.result = FAILURE
212.             }
213.             sendToSubjCoordinator(request)
214.         }
215.     }
216.
217.     //Worker
218.     handleRequest(appRequest request) {
219.         request.result = checkForConstraints(request, database)
220.         //Evaluate conditions based on committed data and tentative data
221.         request.type = POLICY_EVAL_RESULT
222.         sendToSubjCoordinator(request)
223.     }
```