Domain-based system architecture for access control in IOT system using Blockchain

Montdher Alabadi, Oğuz Ata

** E-mail: montdher.alabadi@ogr.altinbas.edu.tr*

Global Contract

```
pragma solidity ^0.4.0;
contract Global{

    constructor () public{
       owner=msg.sender;
   }

address public owner;
  struct GlobalMethods {
    uint ObjResID;
    string ObjResName;
    string ObjResPermision;
    address ObjectAgent;
    address SubjectAgent;
    uint Positivecount;
    uint NegativeCount;
    uint Limit;
    bool Active;
  }
  mapping(uint =>mapping(address => GlobalMethods)) public MethodList;

  modifier OnlyServer {
      require (msg.sender==owner);
      _;
  }
  function RegisterMethod (uint _ObjResID,string memory _ObjResName,
   string memory _ObjResPermision,address _ObjectAgent,address _SubjectAgent,uint _Limit)
OnlyServer public
  {
    if(MethodList[_ObjResID][_SubjectAgent].Active) {revert();
    }
    else
    {
    MethodList[_ObjResID][_SubjectAgent].Active=true;
    MethodList[_ObjResID][_SubjectAgent].ObjResName=_ObjResName;
    MethodList[_ObjResID][_SubjectAgent].ObjResPermision=_ObjResPermision;
    MethodList[_ObjResID][_SubjectAgent].ObjectAgent=_ObjectAgent;
    MethodList[_ObjResID][_SubjectAgent].Positivecount=0;
    MethodList[_ObjResID][_SubjectAgent].NegativeCount=0;
      MethodList[_ObjResID][_SubjectAgent].Limit=_Limit;
     }
  }

  function DeleteMethod(uint _ObjResID,address _SubjectAgent)  OnlyServer
  {
   if(MethodList[_ObjResID][_SubjectAgent].Active){
   delete MethodList[_ObjResID][_SubjectAgent];
```

```
 }}


 function GetMethod (uint _ObjResID,address _SubjectAgent)  OnlyServer public returns
 ( string memory _ObjResPermision, address _ObjectAgent, uint _Positivecount,uint
_NegativeCount,uint _Limit)
 {
  if(MethodList[_ObjResID][_SubjectAgent].Active){
  return(

   _ObjResPermision= MethodList[_ObjResID][_SubjectAgent].ObjResPermision,
    _ObjectAgent=MethodList[_ObjResID][_SubjectAgent].ObjectAgent,
    _Positivecount=MethodList[_ObjResID][_SubjectAgent].Positivecount,
    _NegativeCount=MethodList[_ObjResID][_SubjectAgent].NegativeCount,
     _Limit=MethodList[_ObjResID][_SubjectAgent].Limit  );
 }
 }
 function UpdateMethod (uint _ObjResID,string memory _ObjResName, string memory
_ObjResPermision,address _ObjectAgent,address _SubjectAgent,
  uint _Limit,bool _Active ) OnlyServer
 {
  if(MethodList[_ObjResID][_SubjectAgent].Active) {
  MethodList[_ObjResID][_SubjectAgent].Active=_Active;
  if( !MethodList[_ObjResID][_SubjectAgent].Active){
     DeleteFromCache(_ObjResID,_SubjectAgent);
  }
  MethodList[_ObjResID][_SubjectAgent].ObjResName=_ObjResName;
  MethodList[_ObjResID][_SubjectAgent].ObjResPermision=_ObjResPermision;
  if(!compareStrings(MethodList[_ObjResID][_SubjectAgent].ObjResPermision,"allow")){
     DeleteFromCache(_ObjResID,_SubjectAgent);
  }
  MethodList[_ObjResID][_SubjectAgent].ObjectAgent=_ObjectAgent;
  MethodList[_ObjResID][_SubjectAgent].Limit=_Limit;

   }}

 function DeactivateMethod(uint _ObjResID,address _SubjectAgent)OnlyServer  {
  if(MethodList[_ObjResID][_SubjectAgent].Active){
 MethodList[_ObjResID][_SubjectAgent].Active=false;
 if( !MethodList[_ObjResID][_SubjectAgent].Active){
     DeleteFromCache(_ObjResID,_SubjectAgent);
  } }}


 function activateMethod(uint _ObjResID,address _SubjectAgent) OnlyServer{
  if(!MethodList[_ObjResID][_SubjectAgent].Active){
 MethodList[_ObjResID][_SubjectAgent].Active=true;

 }}

struct Cache {
```

```
  uint ResourceID;
  address SubjectDomainAgent ;
 }

 mapping(address=> Cache) public CacheList;

 function AddToCache (uint _ObjResID,address _SubjectAgent) internal
 {
  if(CacheList[_SubjectAgent].ResourceID==_ObjResID){
      revert();
  }
 else
 {CacheList[_SubjectAgent].ResourceID=_ObjResID;
 }
 }
 function DeleteFromCache (uint _ObjResID,address _SubjectAgent)  internal {
     if(CacheList[_SubjectAgent].ResourceID==_ObjResID){
    delete CacheList[_SubjectAgent].ResourceID;
 }}

 struct Block {
  uint ResourceID;
  address SubjectDomainAgent ;
  uint TimeOfBlock;
  bool Blocked;
 }

mapping(uint =>mapping(address => Block)) internal BlockList;

 function AddToblock (uint _ObjResID,address _SubjectAgent, uint _TimeOfBlock)  public
 {
  if(BlockList[_ObjResID][_SubjectAgent].Blocked){
      revert();
  }
 else
 {
      BlockList[_ObjResID][_SubjectAgent].Blocked=true;
     BlockList[_ObjResID][_SubjectAgent].TimeOfBlock=_TimeOfBlock;
 }}

 function TrigerBlockList (uint _ObjResID,address _SubjectAgent,uint _Time)  internal {
   if( _Time >=(BlockList[_ObjResID][_SubjectAgent].TimeOfBlock + 100)) {
    delete BlockList[_ObjResID][_SubjectAgent];
 }}

 event CheckResult (address _SubjectAgent ,uint _SubjectResourceID,address _ObjectAgent ,uint
_ObjResID,bool Result,string details);

 function GlobalAccessControl (uint _ObjResID,address _SubjectAgent, uint _SubjectResourceID)
public {
      address _ObjectAgent=MethodList[_ObjResID][_SubjectAgent].ObjectAgent;
```

```
      if (MethodList[_ObjResID][_SubjectAgent].Active){

     if (CacheList[_SubjectAgent].ResourceID==_ObjResID){

         emit CheckResult (_SubjectAgent ,_SubjectResourceID,_ObjectAgent ,_ObjResID,true,"Accepted
Using Cache in the global contract");
     }
     else {
         TrigerBlockList(_ObjResID,_SubjectAgent,now);
       if ( BlockList[_ObjResID][_SubjectAgent].Blocked){

          emit CheckResult (_SubjectAgent ,_SubjectResourceID,_ObjectAgent ,_ObjResID,false,"OOPs
Subject IN BlockList in the global contract");
       }
           else {
   if (compareStrings("allow",MethodList[_ObjResID][_SubjectAgent].ObjResPermision)){
    MethodList[_ObjResID][_SubjectAgent].Positivecount++;

if(MethodList[_ObjResID][_SubjectAgent].Positivecount==MethodList[_ObjResID][_SubjectAgent].Li
mit){
        AddToCache(_ObjResID,_SubjectAgent);
        MethodList[_ObjResID][_SubjectAgent].Positivecount=0;
    }
 emit CheckResult (_SubjectAgent ,_SubjectResourceID,_ObjectAgent ,_ObjResID,true,"Accepted using
Global Contract ");
 }
  else   {
      MethodList[_ObjResID][_SubjectAgent].NegativeCount++;

if(MethodList[_ObjResID][_SubjectAgent].NegativeCount==MethodList[_ObjResID][_SubjectAgent].Li
mit){
         AddToblock(_ObjResID,_SubjectAgent,now);
          MethodList[_ObjResID][_SubjectAgent].NegativeCount=0;
     }

     emit CheckResult (_SubjectAgent ,_SubjectResourceID,_ObjectAgent ,_ObjResID,false,"Access
denied using global contract ");
 }}}
      }
     else {
       emit CheckResult (_SubjectAgent ,_SubjectResourceID,_ObjectAgent ,_ObjResID,false,"No such
resource in the system ");
     } }
  function compareStrings (string a, string b) pure public returns (bool){
      return keccak256(a) == keccak256(b);
   }
   function deleteglobalcontract() public{
 if(msg.sender == owner){
 selfdestruct(this);
 }
 }}
```

Local Contract 1

```
pragma solidity ^0.4.0;

contract Local1 {

   constructor () public{
      owner=msg.sender;
   }
    address public owner;
    address public ObjectagentAddress =    0x26f6bf76726ef87ab3b329e3fb242d90045c029d ;
    address public GlobalContractAddress = 0x6d417c78db6decebc3b3075e292fc35515e0972d ;


   struct ResourceStruct {

   uint ObjectResID;
   string ObjResName;
   string ObjResPermision;
   address ObjectAgent ;
   uint StructIndex;
  }

  mapping(uint => ResourceStruct) public ResourceList;
  uint [] private ResourceIndex;


  function ChangeAgent(address _newAgent )OnlyServer {
     ObjectagentAddress=_newAgent;
  }
  modifier OnlyServer {
      require (msg.sender==owner);
      _;
  }

  function CheckResource(uint _ObjResID)  view public   returns(bool )
  {
    if(ResourceIndex.length == 0) return false;
    if (ResourceIndex[ResourceList[_ObjResID].StructIndex] == _ObjResID) {
        return true;
    }
    else return false;
  }

  function RegisterResource (uint _ObjResID,string memory _ObjResName,
   string memory _ObjResPermission) OnlyServer
  {
    if(CheckResource(_ObjResID)) revert();
    ResourceList[_ObjResID].ObjResName = _ObjResName;
```

```
    ResourceList[_ObjResID].ObjResPermision   = _ObjResPermission;
     ResourceList[_ObjResID].ObjectAgent   =ObjectagentAddress ;
    ResourceList[_ObjResID].StructIndex     = ResourceIndex.push(_ObjResID)-1;
  }

  function DeleteResource(uint _ObjResID) public OnlyServer
  {
    if(!CheckResource(_ObjResID)) revert();
    uint ResourceToDelete = ResourceList[_ObjResID].StructIndex;
    uint LastResourceKey = ResourceIndex[ResourceIndex.length-1];
    ResourceIndex[ResourceToDelete] = LastResourceKey;
    ResourceList[LastResourceKey].StructIndex = ResourceToDelete;
    ResourceIndex.length--;
  }

  function FetchResource(uint _ObjResID) internal
    returns(string  memory _ObjResName, string memory _ObjResPermission, address _ObjectAgent)
  {
    if(!CheckResource(_ObjResID)){ revert(); }
    return(
      ResourceList[_ObjResID].ObjResName,
      ResourceList[_ObjResID].ObjResPermision,
      ResourceList[_ObjResID].ObjectAgent);
  }

  function updateResourcePermision(uint _ObjResID,string memory _ObjResPermission) public
OnlyServer
    returns(bool)
  {
    if(!CheckResource(_ObjResID)) revert();
    ResourceList[_ObjResID].ObjResPermision = _ObjResPermission;
    return true;
  }

    event CheckResult (address _SubjectAgent ,uint _SubjectResourceID,address _ObjectAgent ,uint
_ObjResID,bool Result,string details);

  function LocalAccessControl (uint _SubjectResourceID,uint _ObjResID)   public {
      require(msg.sender==ObjectagentAddress);
     bool  resultbool=false;
         if(CheckResource(_SubjectResourceID) && CheckResource(_ObjResID)){
         if (compareStrings(ResourceList[_ObjResID].ObjResPermision,"allow")){
            resultbool=true;
             emit
CheckResult(ObjectagentAddress,_SubjectResourceID,ObjectagentAddress,_ObjResID,resultbool,"OBJE
CT RESOURCE IN THE SAME DOAMIN");
         }
     }
     else {
        Global gb=Global(GlobalContractAddress);
        gb.GlobalAccessControl(_ObjResID,ObjectagentAddress,_SubjectResourceID);
```

```
}}

   function compareStrings (string a, string b) pure public returns (bool){
      return keccak256(a) == keccak256(b);
   }
  function deletelocalContract() public{
 if(msg.sender == owner){
 selfdestruct(this);
 }
 }
}

 contract Global{
   function GlobalAccessControl (uint _ObjResID,address _SubjectAgent, uint _SubjectResourceID) ;
   event CheckResult (address _SubjectAgent ,uint _SubjectResourceID,address _ObjectAgent ,uint
_ObjResID,bool Result,string details);

    }
```

Local Contract 2

```
pragma solidity ^0.4.0;

contract Local2 {

   constructor () public{
      owner=msg.sender;
   }
    address public owner;
    address public ObjectagentAddress =    0x3c8195d260fac96419030c322372020158b7948a ;
    address public GlobalContractAddress = 0x6d417c78db6decebc3b3075e292fc35515e0972d ;


   struct ResourceStruct {

   uint ObjectResID;
   string ObjResName;
   string ObjResPermision;
   address ObjectAgent ;
   uint StructIndex;
  }

  mapping(uint => ResourceStruct) public ResourceList;
  uint [] private ResourceIndex;


  function ChangeAgent(address _newAgent )OnlyServer {
     ObjectagentAddress=_newAgent;
  }
  modifier OnlyServer {
      require (msg.sender==owner);
      _;
  }

  function CheckResource(uint _ObjResID)  view public   returns(bool )
  {
    if(ResourceIndex.length == 0) return false;
    if (ResourceIndex[ResourceList[_ObjResID].StructIndex] == _ObjResID) {
        return true;
    }
    else return false;
  }

  function RegisterResource (uint _ObjResID,string memory _ObjResName,
   string memory _ObjResPermission) OnlyServer
  {
    if(CheckResource(_ObjResID)) revert();
    ResourceList[_ObjResID].ObjResName = _ObjResName;
```

```
    ResourceList[_ObjResID].ObjResPermision   = _ObjResPermission;
     ResourceList[_ObjResID].ObjectAgent   =ObjectagentAddress ;
    ResourceList[_ObjResID].StructIndex     = ResourceIndex.push(_ObjResID)-1;
  }

  function DeleteResource(uint _ObjResID) public OnlyServer
  {
    if(!CheckResource(_ObjResID)) revert();
    uint ResourceToDelete = ResourceList[_ObjResID].StructIndex;
    uint LastResourceKey = ResourceIndex[ResourceIndex.length-1];
    ResourceIndex[ResourceToDelete] = LastResourceKey;
    ResourceList[LastResourceKey].StructIndex = ResourceToDelete;
    ResourceIndex.length--;
  }

  function FetchResource(uint _ObjResID) internal
    returns(string  memory _ObjResName, string memory _ObjResPermission, address _ObjectAgent)
  {
    if(!CheckResource(_ObjResID)){ revert(); }
    return(
      ResourceList[_ObjResID].ObjResName,
      ResourceList[_ObjResID].ObjResPermision,
      ResourceList[_ObjResID].ObjectAgent);
  }

  function updateResourcePermision(uint _ObjResID,string memory _ObjResPermission) public
OnlyServer
    returns(bool)
  {
    if(!CheckResource(_ObjResID)) revert();
    ResourceList[_ObjResID].ObjResPermision = _ObjResPermission;
    return true;
  }

    event CheckResult (address _SubjectAgent ,uint _SubjectResourceID,address _ObjectAgent ,uint
_ObjResID,bool Result,string details);

  function LocalAccessControl (uint _SubjectResourceID,uint _ObjResID)   public {
      require(msg.sender==ObjectagentAddress);
     bool  resultbool=false;
         if(CheckResource(_SubjectResourceID) && CheckResource(_ObjResID)){
         if (compareStrings(ResourceList[_ObjResID].ObjResPermision,"allow")){
            resultbool=true;
             emit
CheckResult(ObjectagentAddress,_SubjectResourceID,ObjectagentAddress,_ObjResID,resultbool,"OBJE
CT RESOURCE IN THE SAME DOAMIN");
         }
     }
     else {
        Global gb=Global(GlobalContractAddress);
        gb.GlobalAccessControl(_ObjResID,ObjectagentAddress,_SubjectResourceID);
```

```
 }}

    function compareStrings (string a, string b) pure public returns (bool){
       return keccak256(a) == keccak256(b);
    }
   function deletelocalContract() public{
 if(msg.sender == owner){
 selfdestruct(this);
 }
 }
}

 contract Global{
    function GlobalAccessControl (uint _ObjResID,address _SubjectAgent, uint _SubjectResourceID) ;
    event CheckResult (address _SubjectAgent ,uint _SubjectResourceID,address _ObjectAgent ,uint
_ObjResID,bool Result,string details);

     }
```