Contract Patterns and Security

Presented by
Anthony C. Eufemio
Chief Technology Officer @ DigixGlobal
ace@dgx.io

Key Points

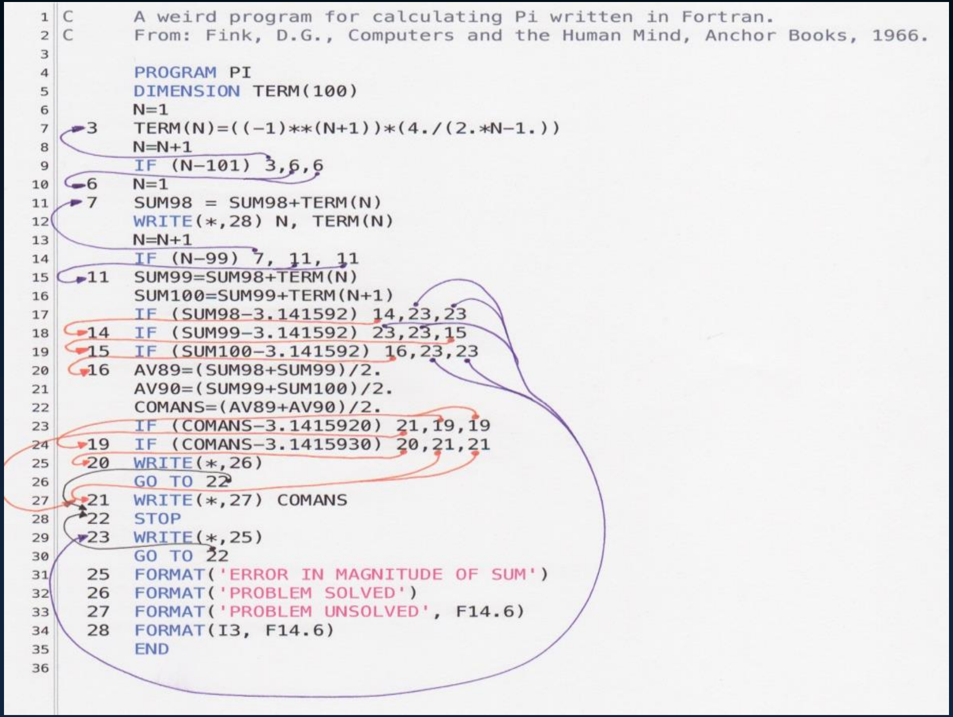
1. Solidity is not broken!
2. Let’s use software architecture as an important component in writing secure Ethereum contracts.
3. Use code patterns that allow us to minimize the attack surface.
4. Understanding Contract Execution Flow.
5. Case Study: DigixCore Architecture

Bold Statement of the Day: Good Software Architecture Leads to Good Security

Good software design patterns gives us:

1. Code readability.

Spaghetti Code



```
C     A weird program for calculating Pi written in Fortran.
C     From: Fink, D.G., Computers and the Human Mind, Anchor Books, 1966.

      PROGRAM PI
      DIMENSION TERM(100)
      N=1
    3 TERM(N)=((-1)**(N+1))*(4./(2.*N-1.))
      N=N+1
      IF (N-101) 3,6,6
    6 N=1
    7 SUM98 = SUM98+TERM(N)
      WRITE(*,28) N, TERM(N)
      N=N+1
      IF (N-99) 7, 11, 11
   11 SUM99=SUM98+TERM(N)
      SUM100=SUM99+TERM(N+1)
      IF (SUM98-3.141592) 14,23,23
   14 IF (SUM99-3.141592) 23,23,15
   15 IF (SUM100-3.141592) 16,23,23
   16 AV89=(SUM98+SUM99)/2.
      AV90=(SUM99+SUM100)/2.
      COMANS=(AV89+AV90)/2.
      IF (COMANS-3.1415920) 21,19,19
   19 IF (COMANS-3.1415930) 20,21,21
   20 WRITE(*,26)
      GO TO 22
   21 WRITE(*,27) COMANS
   22 STOP
   23 WRITE(*,25)
      GO TO 22
   25 FORMAT('ERROR IN MAGNITUDE OF SUM')
   26 FORMAT('PROBLEM SOLVED')
   27 FORMAT('PROBLEM UNSOLVED', F14.6)
   28 FORMAT(I3, F14.6)
      END
```

Bold Statement of the Day: Good Software Architecture Leads to Good Security

Good software design patterns gives us:

1. Code readability.
2. Re-usable “battle tested” code.

Bold Statement of the Day: Good Software Architecture Leads to Good Security

Good software design patterns gives us:

1. Code readability.
2. Re-usable battle tested code.
3. Separation of concerns.

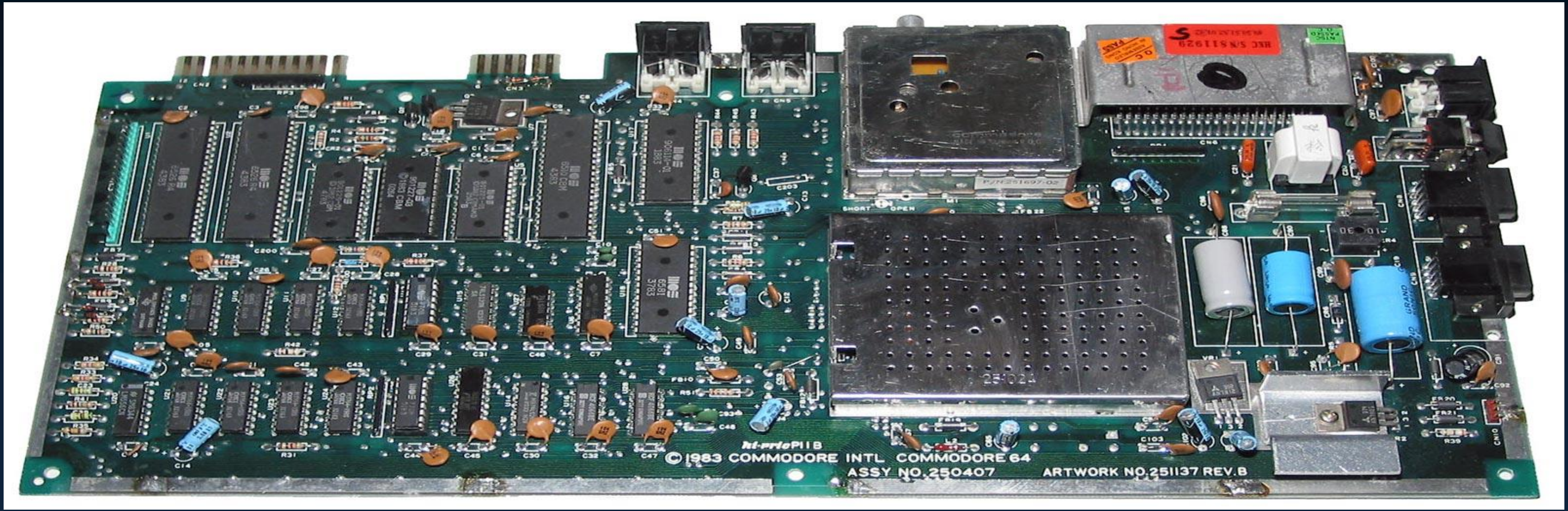
Electronic Circuits vs. Code

- Contracts deployed on Ethereum, once deployed, cannot be changed

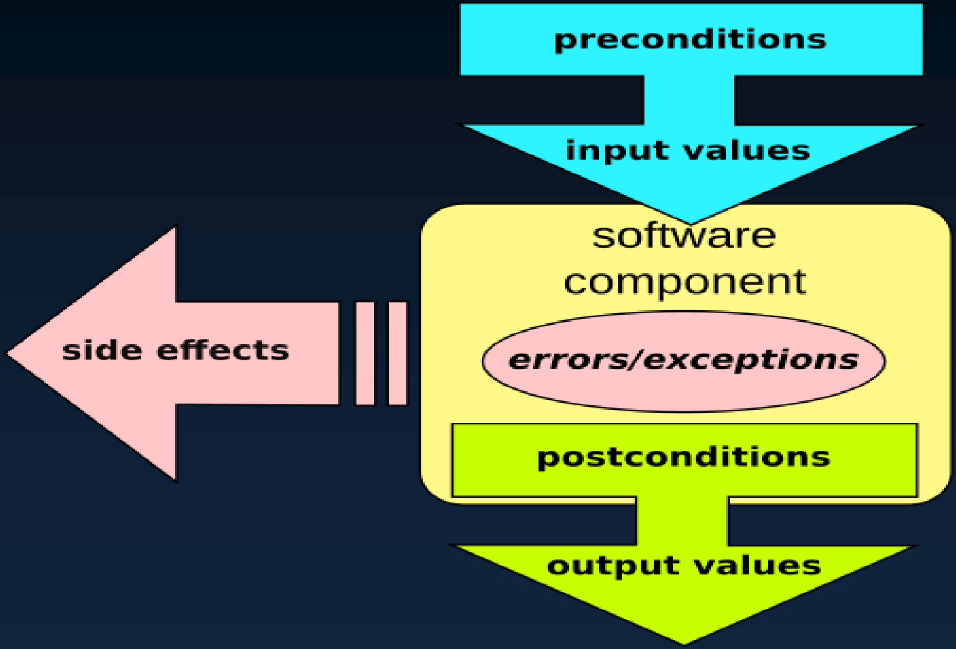
Electronic Circuits vs. Code

- Contracts deployed on Ethereum cannot be changed
- Electronic circuit boards vs. software code

Electronic Circuit Board



Design By Contract



Electronic Circuits vs. Code

- Contracts deployed on Ethereum cannot be changed
- Electronic circuit boards vs. software code

Bold Statement of the Day: Good Software Architecture Leads to Good Security

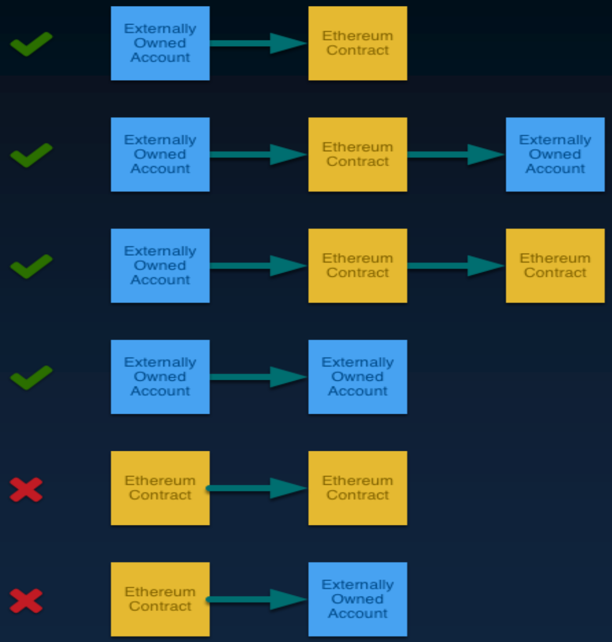
Good software design patterns gives us:

1. Code readability.
2. Re-usable battle tested code.
3. Separation of concerns.
4. Better opportunities for tooling (i.e. static analysis)

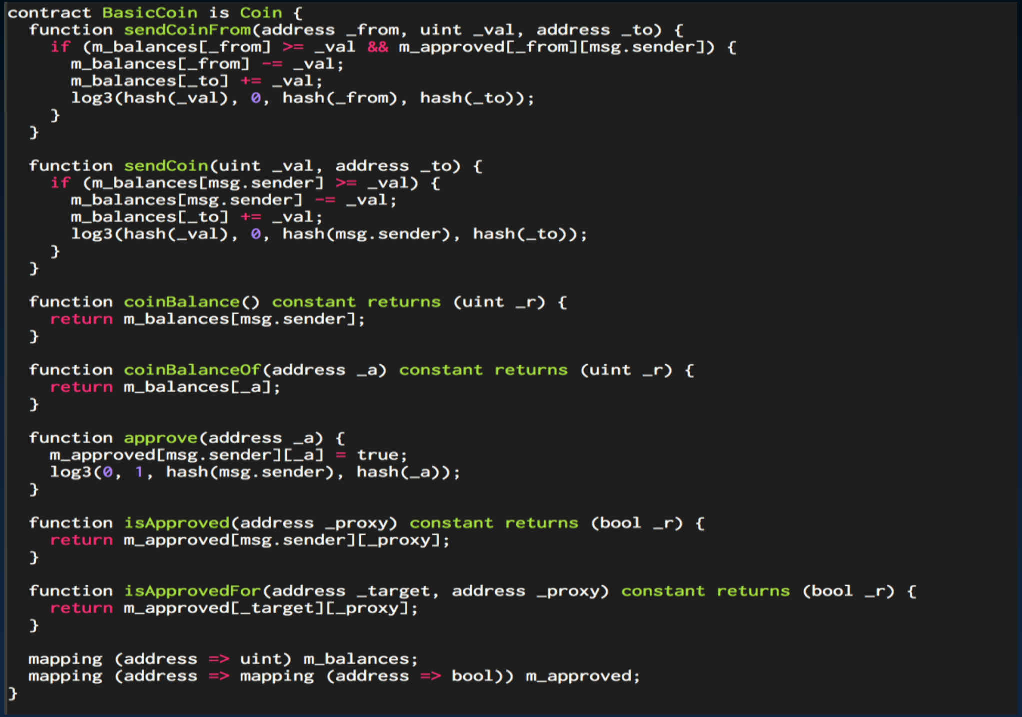
Attack Vectors

- Attack vectors are code paths that can allow a malicious user to gain unauthorized access to a software or computer resource.
  - Command line arguments.
  - Environment variables
  - URL parameters and HTTP POST data.
  - Data storage.
  - Function parameters/3rd Party Contracts

Ethereum TX


✓ Externally Owned Account → Ethereum Contract
✓ Externally Owned Account → Ethereum Contract → Externally Owned Account
✓ Externally Owned Account → Ethereum Contract → Ethereum Contract
✓ Externally Owned Account → Externally Owned Account
✗ Ethereum Contract → Ethereum Contract
✗ Ethereum Contract → Externally Owned Account

Example ERC-20 Token



```
contract BasicCoin is Coin {
  function sendCoinFrom(address _from, uint _val, address _to) {
    if (m_balances[_from] >= _val && m_approved[_from][msg.sender]) {
      m_balances[_from] -= _val;
      m_balances[_to] += _val;
      log3(hash(_val), 0, hash(_from), hash(_to));
    }
  }

  function sendCoin(uint _val, address _to) {
    if (m_balances[msg.sender] >= _val) {
      m_balances[msg.sender] -= _val;
      m_balances[_to] += _val;
      log3(hash(_val), 0, hash(msg.sender), hash(_to));
    }
  }

  function coinBalance() constant returns (uint _r) {
    return m_balances[msg.sender];
  }

  function coinBalanceOf(address _a) constant returns (uint _r) {
    return m_balances[_a];
  }

  function approve(address _a) {
    m_approved[msg.sender][_a] = true;
    log3(0, 1, hash(msg.sender), hash(_a));
  }

  function isApproved(address _proxy) constant returns (bool _r) {
    return m_approved[msg.sender][_proxy];
  }

  function isApprovedFor(address _target, address _proxy) constant returns (bool _r) {
    return m_approved[_target][_proxy];
  }

  mapping (address => uint) m_balances;
  mapping (address => mapping (address => bool)) m_approved;
}
```

ICS Pattern

Let us can break down our contract into separate contracts based on their specific behaviors

- Interface Contracts
- Controller Contracts
- Storage Contracts
- Access Control (CACP)
- Directory Service/Resolver

Interface Contracts

Contracts that can be directly called by an end-user or external contracts. This is the main entry point.

Controller Contracts

Contracts that can be called from interface contracts or other controller contracts that performs a set of tasks based on the input from the calling contracts.

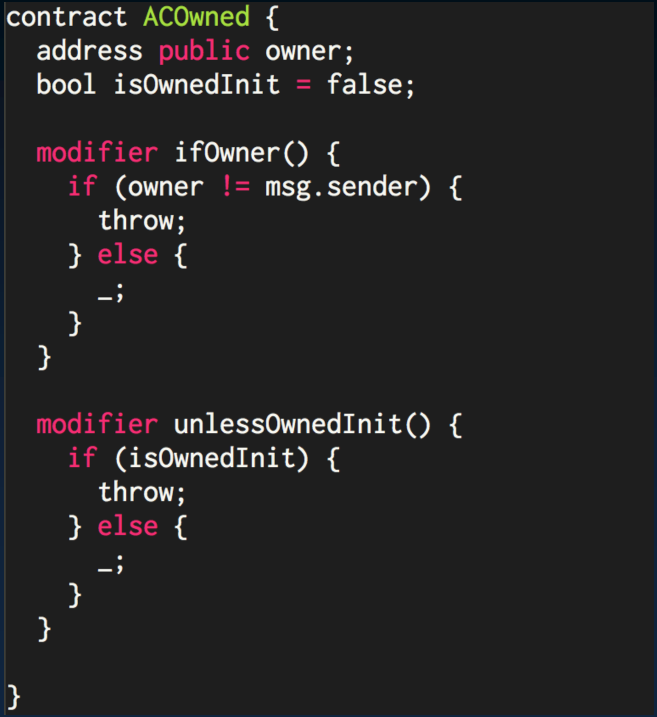
Storage Contracts

Contracts that store data (i.e. state information) which can be called by controller contracts.

Other Components

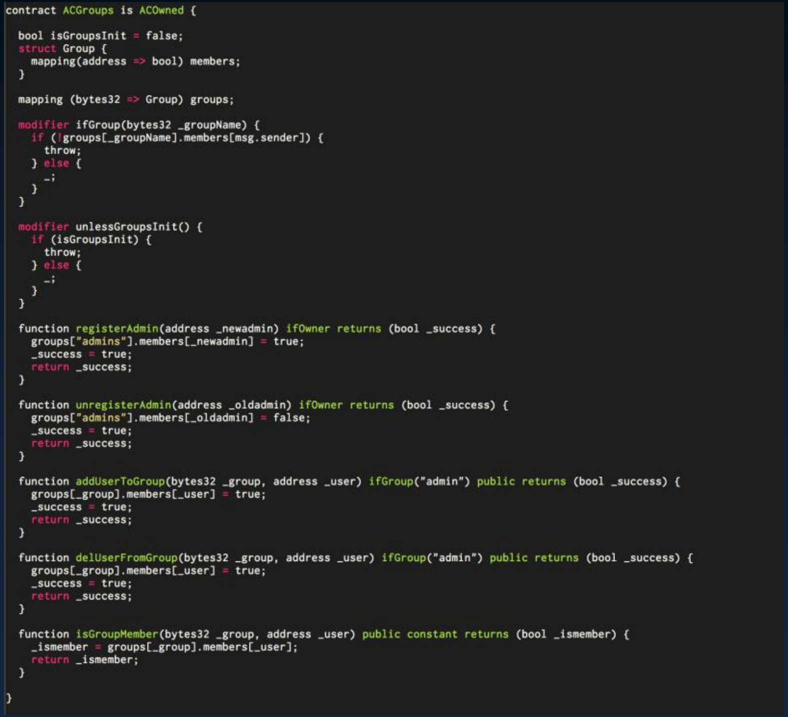
Access Control – a set of re-usable code that we can use to describe how a resource can be accessed.

Access Control



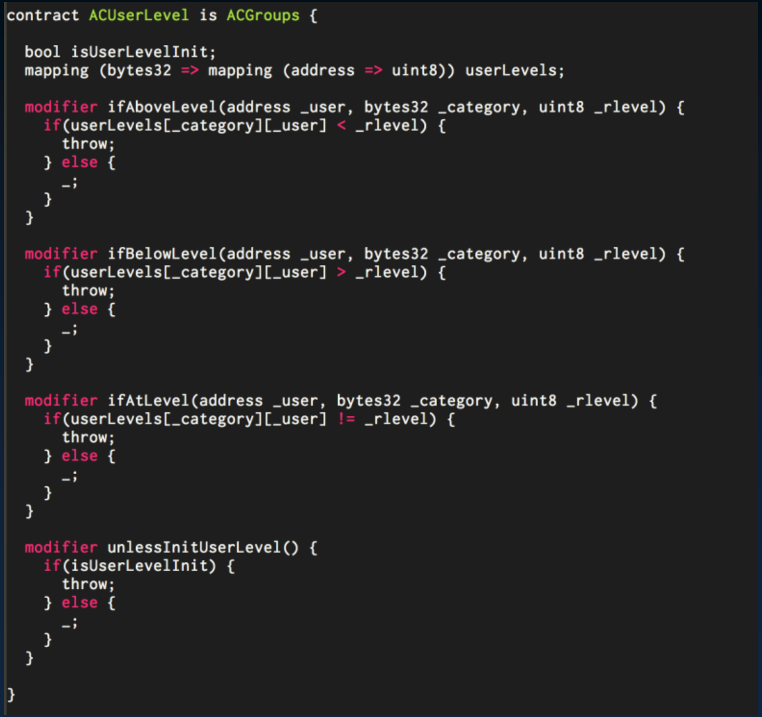
```
contract ACOwned {
  address public owner;
  bool isOwnedInit = false;

  modifier ifOwner() {
    if (owner != msg.sender) {
      throw;
    } else {
      _;
    }
  }

  modifier unlessOwnedInit() {
    if (isOwnedInit) {
      throw;
    } else {
      _;
    }
  }

}
```

Access Control



```
contract ACGroups is ACOwned {

  bool isGroupsInit = false;
  struct Group {
    mapping(address => bool) members;
  }

  mapping (bytes32 => Group) groups;

  modifier ifGroup(bytes32 _groupName) {
    if (!groups[_groupName].members[msg.sender]) {
      throw;
    } else {
      _;
    }
  }

  modifier unlessGroupsInit() {
    if (isGroupsInit) {
      throw;
    } else {
      _;
    }
  }

  function registerAdmin(address _newadmin) ifOwner returns (bool _success) {
    groups["admins"].members[_newadmin] = true;
    _success = true;
    return _success;
  }

  function unregisterAdmin(address _oldadmin) ifOwner returns (bool _success) {
    groups["admins"].members[_oldadmin] = false;
    _success = true;
    return _success;
  }

  function addUserToGroup(bytes32 _group, address _user) ifGroup("admin") public returns (bool _success) {
    groups[_group].members[_user] = true;
    _success = true;
    return _success;
  }

  function delUserFromGroup(bytes32 _group, address _user) ifGroup("admin") public returns (bool _success) {
    groups[_group].members[_user] = true;
    _success = true;
    return _success;
  }

  function isGroupMember(bytes32 _group, address _user) public constant returns (bool _ismember) {
    _ismember = groups[_group].members[_user];
    return _ismember;
  }

}
```

Access Control



```
contract ACUserLevel is ACGroups {

  bool isUserLevelInit;
  mapping (bytes32 => mapping (address => uint8)) userLevels;

  modifier ifAboveLevel(address _user, bytes32 _category, uint8 _rlevel) {
    if(userLevels[_category][_user] < _rlevel) {
      throw;
    } else {
      _;
    }
  }

  modifier ifBelowLevel(address _user, bytes32 _category, uint8 _rlevel) {
    if(userLevels[_category][_user] > _rlevel) {
      throw;
    } else {
      _;
    }
  }

  modifier ifAtLevel(address _user, bytes32 _category, uint8 _rlevel) {
    if(userLevels[_category][_user] != _rlevel) {
      throw;
    } else {
      _;
    }
  }

  modifier unlessInitUserLevel() {
    if(isUserLevelInit) {
      throw;
    } else {
      _;
    }
  }

}
```

Other Components

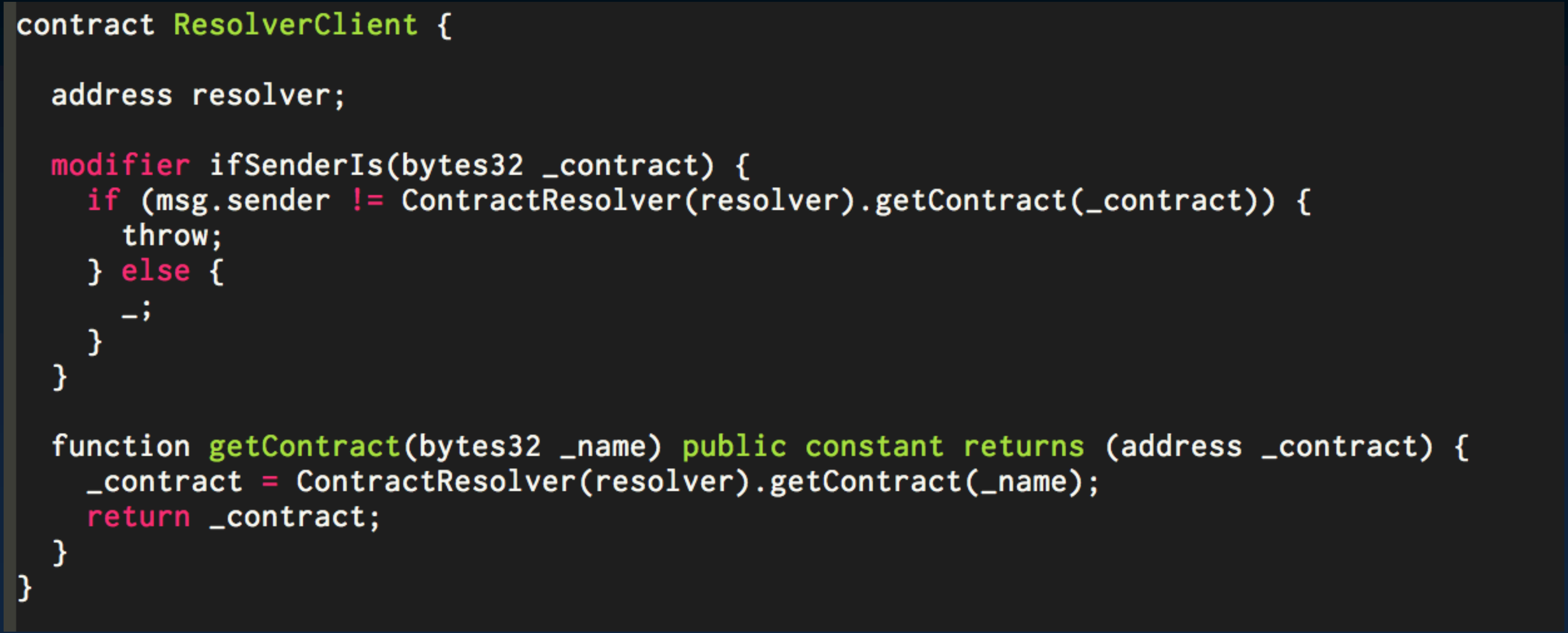
Access Control – a set of re-usable code that we can use to describe how a resource can be accessed.

Directory Service – key value store that allows interface and controller contracts to call other contracts.

Directory Service

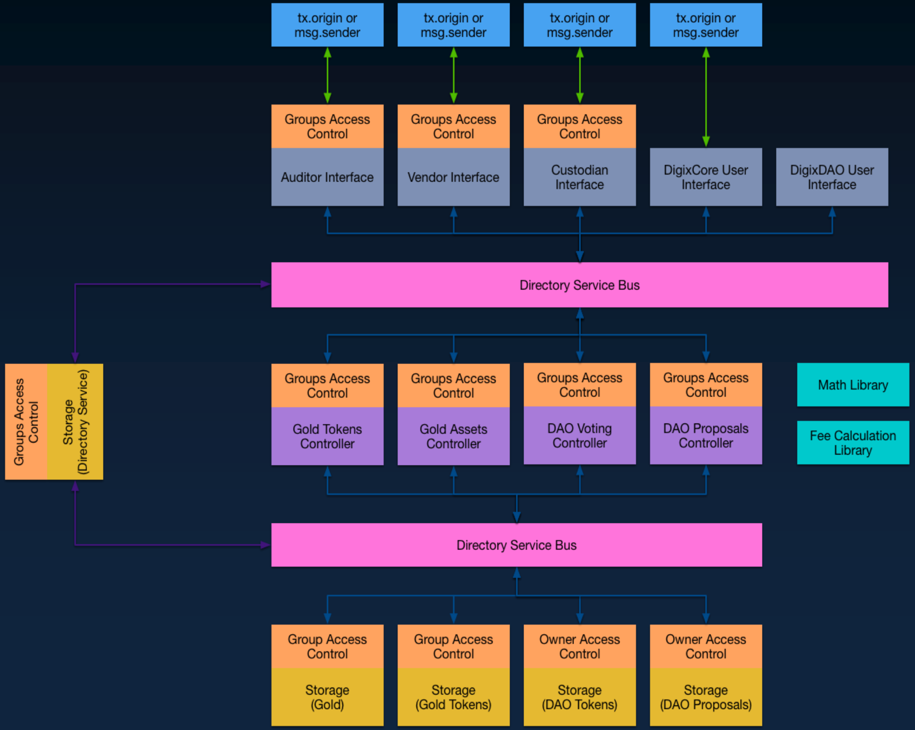
```
import "./AC.sol";

contract ContractResolver is ACGroups {

  mapping (bytes32 => address) contracts;

  event RegisterEvent(bytes32 indexed _contractName, address indexed _contractAddress);

  function ContractResolver() {
    owner = msg.sender;
    groups["admins"].members[msg.sender] = true;
    groups["nsadmins"].members[msg.sender] = true;
  }

  function registerContract(bytes32 _name, address _contract) ifGroup("nsadmins") returns (bool _success) {
    contracts[_name] = _contract;
    _success = true;
    RegisterEvent(_name, _contract);
    return _success;
  }

  function getContract(bytes32 _name) public constant returns (address _contract) {
    _contract = contracts[_name];
    return _contract;
  }

}
```

Directory Service Resolver



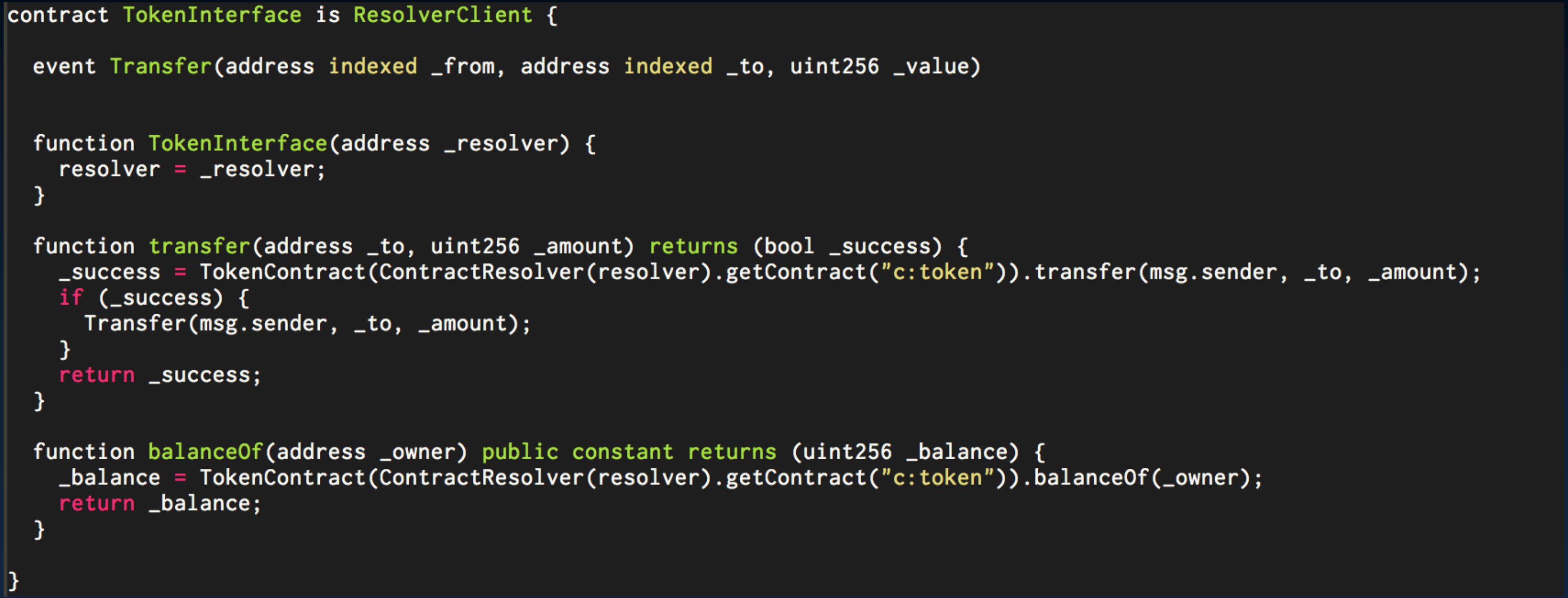
```
contract ResolverClient {

  address resolver;

  modifier ifSenderIs(bytes32 _contract) {
    if (msg.sender != ContractResolver(resolver).getContract(_contract)) {
      throw;
    } else {
      _;
    }
  }

  function getContract(bytes32 _name) public constant returns (address _contract) {
    _contract = ContractResolver(resolver).getContract(_name);
    return _contract;
  }
}
```

ICS Pattern



tx.origin or msg.sender
tx.origin or msg.sender
tx.origin or msg.sender
tx.origin or msg.sender
Groups Access Control
Auditor Interface
Groups Access Control
Vendor Interface
Groups Access Control
Custodian Interface
DigixCore User Interface
DigixDAO User Interface
Directory Service Bus
Groups Access Control
Storage (Directory Service)
Groups Access Control
Gold Tokens Controller
Groups Access Control
Gold Assets Controller
Groups Access Control
DAO Voting Controller
Groups Access Control
DAO Proposals Controller
Math Library
Fee Calculation Library
Directory Service Bus
Group Access Control
Storage (Gold)
Group Access Control
Storage (Gold Tokens)
Owner Access Control
Storage (DAO Tokens)
Owner Access Control
Storage (DAO Proposals)

Interface Contract Example

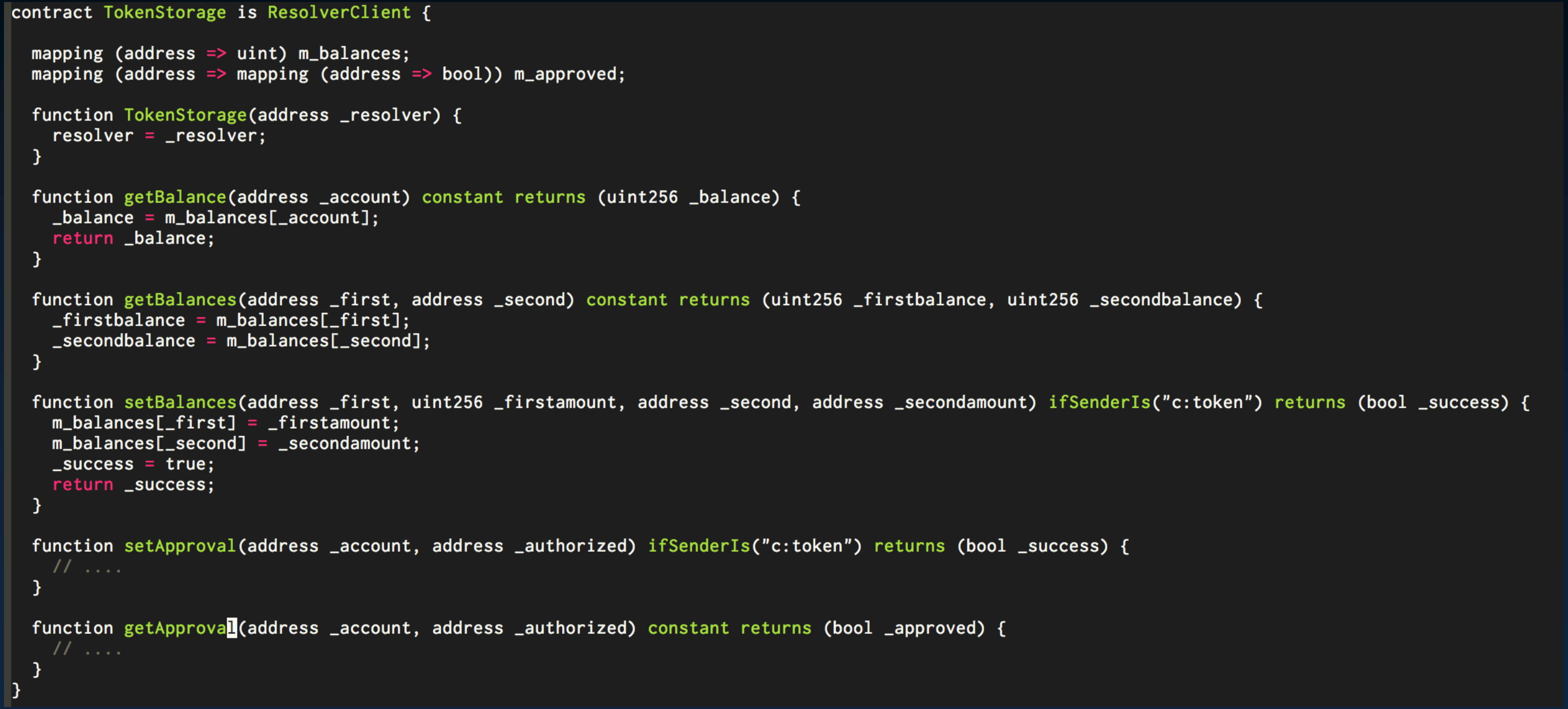


```
contract TokenInterface is ResolverClient {

  event Transfer(address indexed _from, address indexed _to, uint256 _value)


  function TokenInterface(address _resolver) {
    resolver = _resolver;
  }

  function transfer(address _to, uint256 _amount) returns (bool _success) {
    _success = TokenContract(ContractResolver(resolver).getContract("c:token")).transfer(msg.sender, _to, _amount);
    if (_success) {
      Transfer(msg.sender, _to, _amount);
    }
    return _success;
  }

  function balanceOf(address _owner) public constant returns (uint256 _balance) {
    _balance = TokenContract(ContractResolver(resolver).getContract("c:token")).balanceOf(_owner);
    return _balance;
  }

}
```

Controller Contract Example

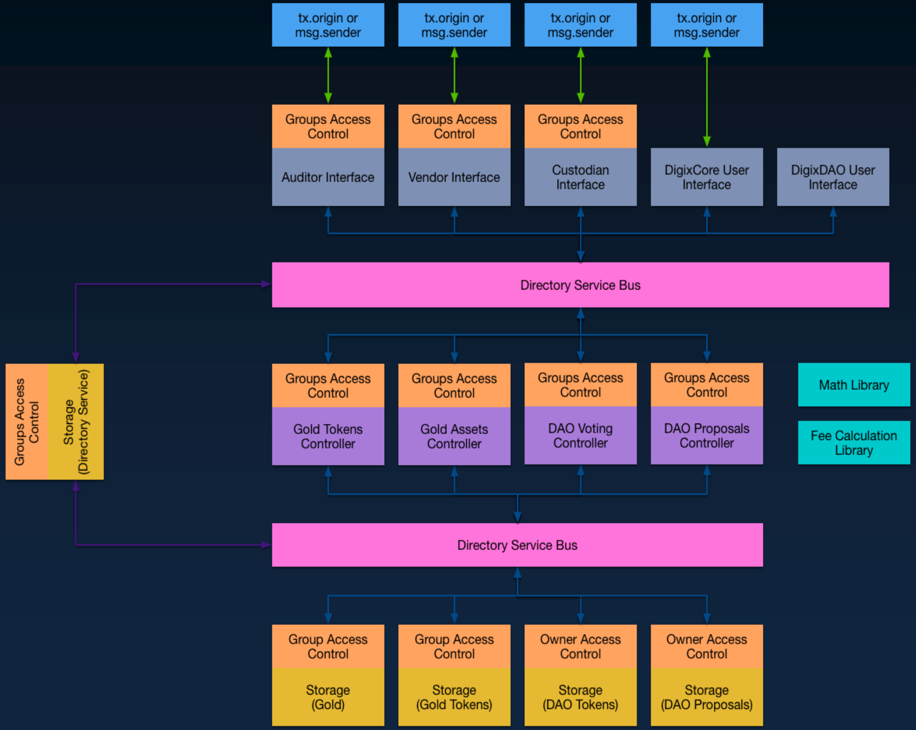
```
contract TokenController is ResolverClient {

  function TokenController(address _resolver) {
    resolver = _resolver;
  }

  function transfer(address _from, address _to, uint256 _amount) ifSenderIs(”i:token”) returns (bool _success) {
    _firstbalance, _secondbalance = TokenContract(ContractResolver(resolver).getContract(”s:token”)).getBalances(_from, _to);
    _firstnewbalance = DigixMath.safeSubtract(_from, _amount);
    _secondnewbalance = DigixMath.safeAdd(_to, _amount);
    _success = TokenContract(ContractResolver(resolver).getContract(”s:token”)).setBalances(_from, _firstnewbalance, _to, _secondnewbalance);
    return _success;
  }

  function balanceOf(address _account) constant returns (uint256 _balance) {
    _balance = TokenContract(ContractResolver(resolver).getContract(”s:token”)).getBalance(_account);
    return _balance;
  }

}
```

Storage Contract Example



```
contract TokenStorage is ResolverClient {

  mapping (address => uint) m_balances;
  mapping (address => mapping (address => bool)) m_approved;

  function TokenStorage(address _resolver) {
    resolver = _resolver;
  }

  function getBalance(address _account) constant returns (uint256 _balance) {
    _balance = m_balances[_account];
    return _balance;
  }

  function getBalances(address _first, address _second) constant returns (uint256 _firstbalance, uint256 _secondbalance) {
    _firstbalance = m_balances[_first];
    _secondbalance = m_balances[_second];
  }

  function setBalances(address _first, uint256 _firstamount, address _second, address _secondamount) ifSenderIs("c:token") returns (bool _success) {
    m_balances[_first] = _firstamount;
    m_balances[_second] = _secondamount;
    _success = true;
    return _success;
  }

  function setApproval(address _account, address _authorized) ifSenderIs("c:token") returns (bool _success) {
    // ....
  }

  function getApproval(address _account, address _authorized) constant returns (bool _approved) {
    // ....
  }
}
```

ICS Pattern



tx.origin or msg.sender
tx.origin or msg.sender
tx.origin or msg.sender
tx.origin or msg.sender
Groups Access Control
Auditor Interface
Groups Access Control
Vendor Interface
Groups Access Control
Custodian Interface
DigixCore User Interface
DigixDAO User Interface
Directory Service Bus
Groups Access Control
Storage (Directory Service)
Groups Access Control
Gold Tokens Controller
Groups Access Control
Gold Assets Controller
Groups Access Control
DAO Voting Controller
Groups Access Control
DAO Proposals Controller
Math Library
Fee Calculation Library
Directory Service Bus
Group Access Control
Storage (Gold)
Group Access Control
Storage (Gold Tokens)
Owner Access Control
Storage (DAO Tokens)
Owner Access Control
Storage (DAO Proposals)

Closing

- Conclusions
- Reach me on:
  - Digix Public Slack https://dgx.io
  - Reddit /u/aedigix
  - Twitter @tym4t
- DigixGlobal @ Demo Day