

COMP6452

Software Architecture for Blockchain Applications

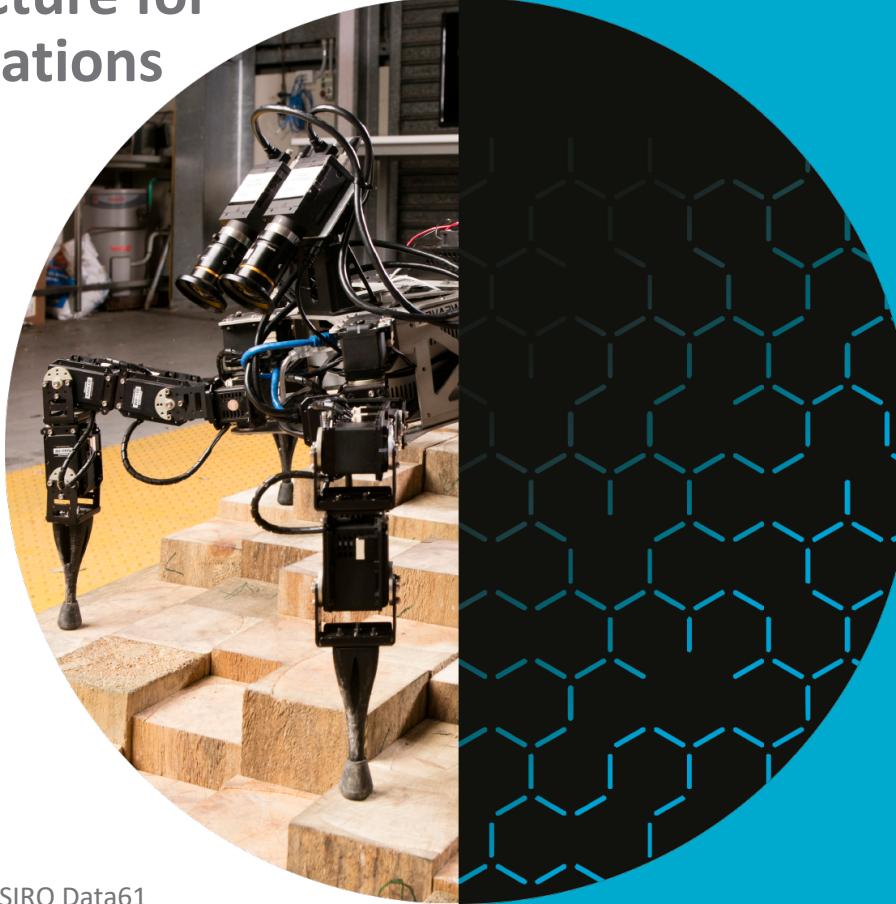
Model- driven Engineering

Helen Paik

| Senior Lecturer @ CSE, UNSW

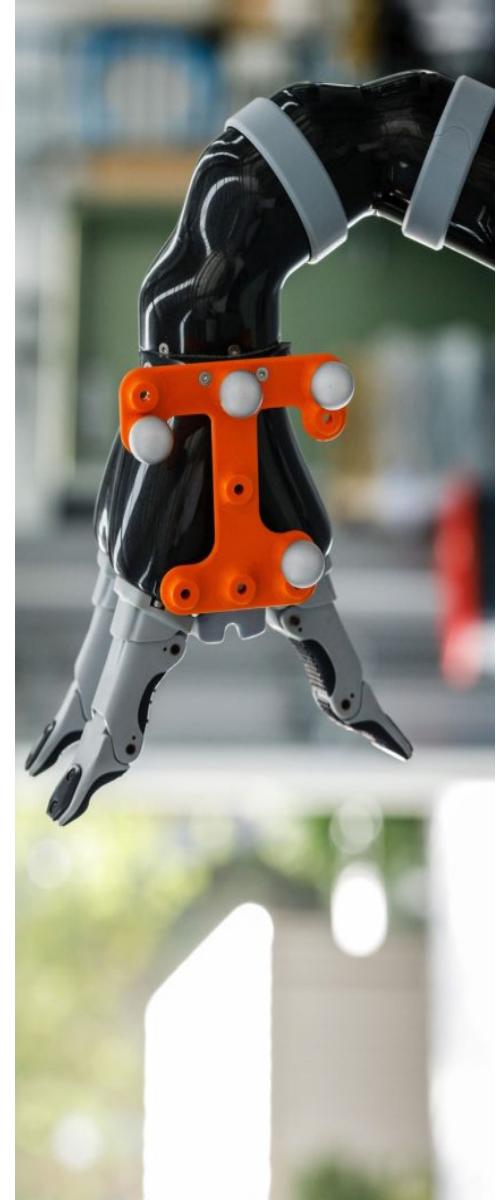
| Visiting Researcher @ AAP team, CSIRO Data61

h.paik@unsw.edu.au



Outline

- Model-Driven Engineering basics
- MDE for data and token model
- Business Process Model and Notation (BPMN)
- MDE for process models





Model-driven Engineering Basics

Motivation for MDE

- Blockchain is still a relatively new technology with limited tooling and documentation, skill set is rare
- According to a survey by Gartner, “23% of [relevant surveyed] CIOs said that blockchain requires the newest skills for implementation at various application areas, while 18% said that blockchain technique itself is one of the most difficult.”¹
- Mistakes in smart contracts have led to massive economic loss, such as the DAO exploit.
- According to an ACS / Data61 report from 2019: 14 jobs per blockchain developer²

[1] Gartner Press Release. Gartner Survey Reveals the Scarcity of Current Blockchain Deployments. (3 May 2018).
<https://www.gartner.com/newsroom/id/3873790>

[2] Bratanova, A., Devaraj, D., Horton, J., Naughtin, C., Kloester, B., Trinh, K., Weber, I., Dawson, D. (2019) Blockchain 2030: A Look at the Future of Blockchain in Australia. CSIRO Data61: Brisbane, Australia.
https://www.researchgate.net/publication/332298704_Blockchain_2030_A_Look_at_the_Future_of_Blockchain_in_Australia

Model-driven Engineering

- A methodology for using models at various levels of abstraction and for different purposes during software development
 - Often concern-specific or domain-specific modelling languages
 - Text-based or graphical modelling notations can be used (i.e., not *necessarily* graphical)
 - Can cover static structures (like data models) or dynamic behaviour (like activity sequences)
- Requires you to formalize the architecture, can be an “architectural catalyst”
 - Helps to get a usable blueprint of the system
 - Makes architecture models immediately useful & motivates up-to-dateness
 - Architecture diagrams become **technical artefacts** of the system
- MDE can increase code quality and, even when starting from scratch, pay off within weeks

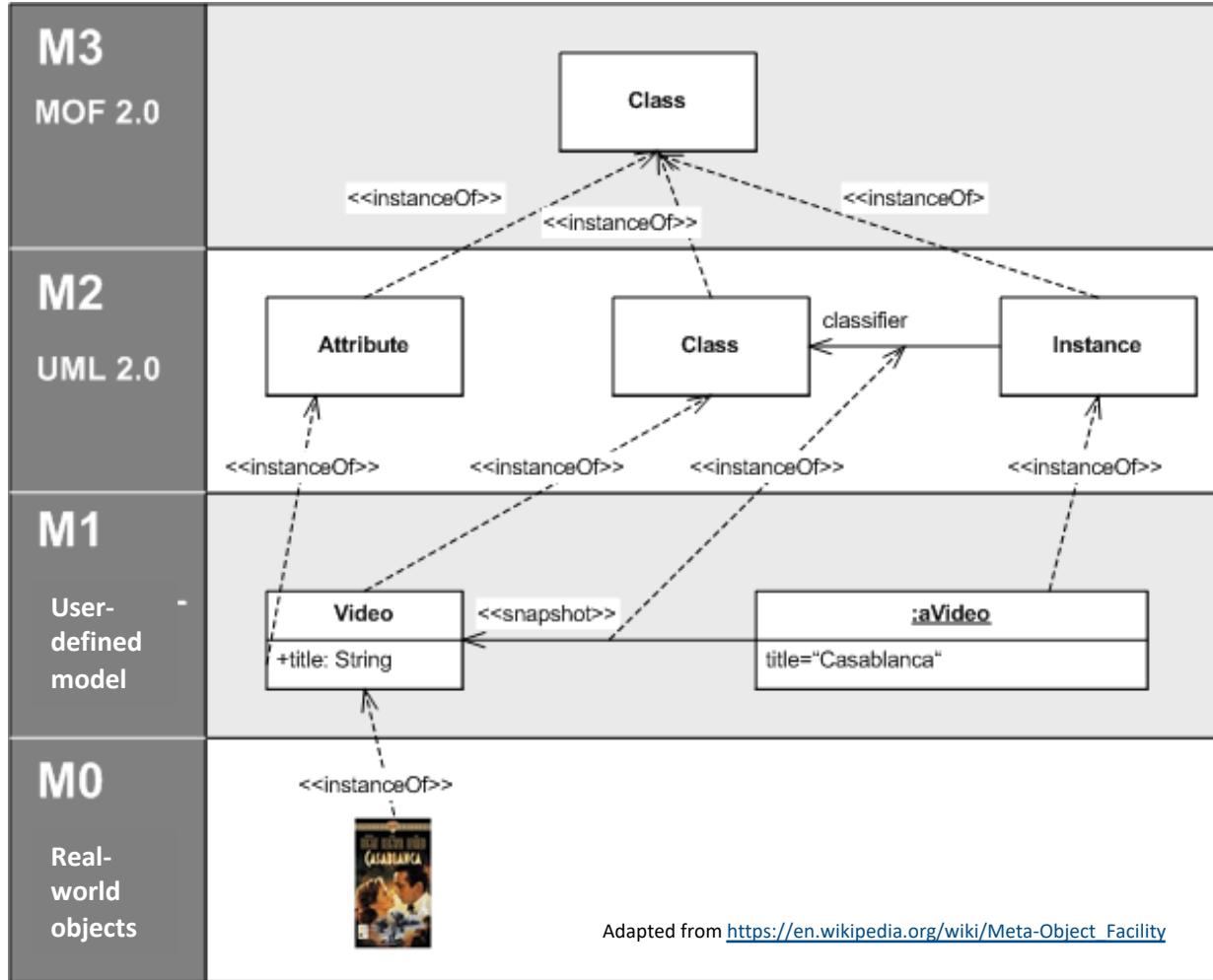
Model-driven Engineering – modelling levels

- Models are abstractions; what to model and what to abstract depends on the purpose
 - “[A] model which took account of all the variegation of reality would be of no more use than a map at the scale of one to one.” Joan Robinson, 1962
- Low-level models: production code can be directly derived from the models
- High-level models: means of communication between business owners and developers implementing a system
- Intermediate levels can support model-based system analysis or system management tools
- Any level: can generate some of the code:
 - Code skeleton or early version of the code
 - Glue code and boilerplate code

MDE vs. MDD vs. MDA

- Model-driven development (MDD) vs. MDE:
 - The parts of MDE that focus on code generation
 - Think of software engineering vs. software development
- Model-driven architecture (MDA) vs. MDD/MDE:
 - OMG's standard for MDE, using OMG modelling languages – primarily Unified Modeling Language (UML)
 - <https://www.omg.org/mda/>
 - Does not cover all aspects of an architecture, but architecture-centric models
 - Foundation standard: Meta-Object Facility (MOF) – see next slide
 - Other prominent concepts:
 - Platform-Independent Model (PIM)
 - Platform-Specific Model (PSM)
 - Transformations: model-to-model or model-to-code

Meta-Object Facility (MOF)



Advantages of MDE in the blockchain context

- Code generation can implement best practices and well-tested building blocks
 - Code can adhere to blockchain “standards” (like ERC-20, ERC-721, ...)
- Improved productivity, especially for novices in a particular technology
 - However: someone should understand the code and review it
- Models can be independent of specific blockchain technologies or platforms
 - Avoid lock-in to specific blockchain technologies
- Models are often easier to understand than code – particularly useful in communicating with business partners about smart contracts
 - Facilitates building trust
 - Domain experts can understand how their ideas are represented in the system
 - Business owners can better assess if smart contract (from a model) is acceptable



MDE for Data and Token Model

Fungible and Non-fungible Tokens

Recap from previous lectures

- Fungible tokens:
interchangeable
 - E.g., \$2 coin, \$10 note
 - Main concern: how many?
 - Ethereum: ERC20 standard
 - https://theethereum.wiki/w/index.php/ERC20_Token_Standard
 - Example: OmiseGO (OMG).

“The OmiseGO blockchain comprises a decentralized exchange, liquidity provider mechanism, clearinghouse messaging network, and asset-backed blockchain gateway. ... It uses the mechanism of a protocol token to create a proof-of-stake blockchain to enable enforcement of market activity amongst participants. Owning OMG tokens buys the right to validate this blockchain, within its consensus rules.”
- Non-fungible tokens
 - E.g., houses, cars, patents
 - Main concern: which ones?
 - Ethereum: ERC721
 - <https://github.com/ethereum/EIPs/issues/721>
 - Example: cryptokitties
<https://www.cryptokitties.co/>
 - Kitties are non-fungible, individual, and their appearance depends on the individual features



MDE for data structures and tokens

- Approach:
 - Enter high-level information
 - Model data structure (variables, types) – not for fungible tokens
 - Model relationships to other types / tokens
 - Select features
- Code is generated – deploy directly or customize
- Feature examples:
 - Fungible tokens:
 - Can be minted? Burnt? By whom?
 - Non-fungible tokens
 - Include standard method(s) for sale
 - One contract for all tokens or one per token?
- Code generated is compliant with standards
 - interface syntax and semantics

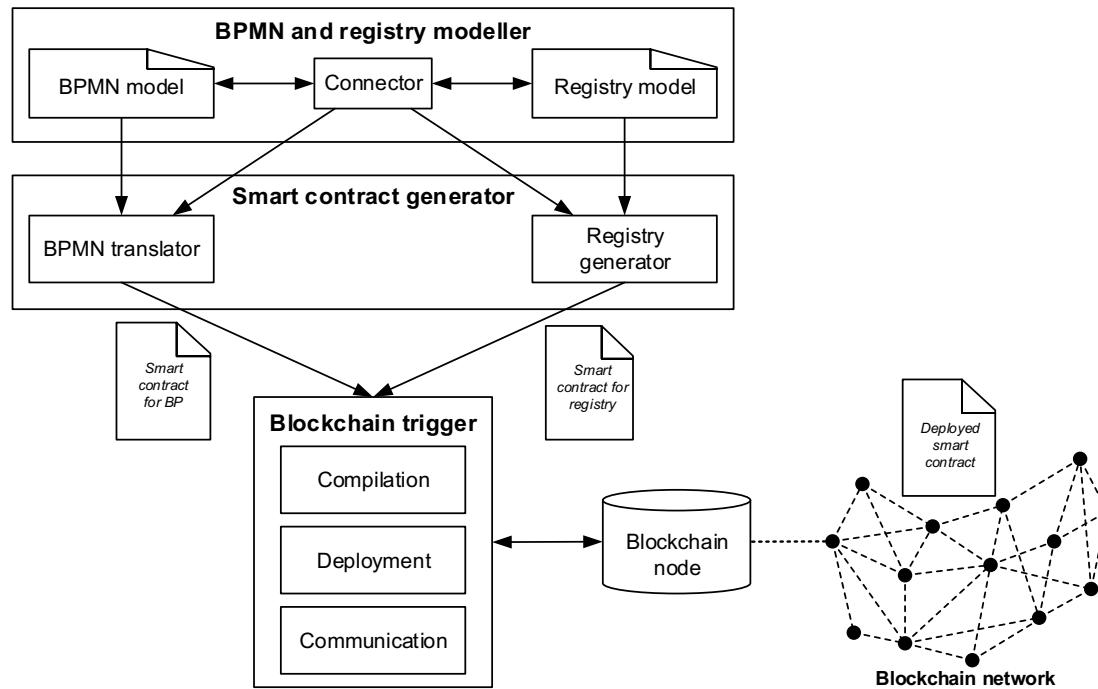


Student Task

- Pause for a second ... and:
 - Think about what data to model for students of this class
 - Write down your top 4-10 data fields
 - Think about any class-specific functions that might be relevant
 - Note: the tool may not support, but still worth writing them down

Data61 tool: Lorikeet

- Lorikeet: automatic generate smart contracts from BPMN models/registry data schema



Edit Registry Design

Compile Smart Contract Save & Close Cancel

Digital Asset Name: COMP6452-students
The digital assets to be registered on the blockchain.

Registry Type: Single Distributed
All records held in a single registry smart contract.

Record ID Data Type: address
Unique ID identifying each registry record.

Record Attributes

New Attribute: string StudentName
 Allow updates to this attribute Add Attribute

Data Type	Attribute Name
uint	z-ID

Foreign Keys

Referenced Registry: This registry (self-referenced)
Create a reference from an attribute in this registry to the record ID(s) of this or another registry.

Foreign Key Attribute: address FK Name
 Many-to-One Many-to-Many Add Foreign Key

Smart Contract Output

Source Code ▾

```
pragma solidity ^ 0.4 .21;

// COMP6452-students registry smart contract.
contract COMP6452 - studentsRegistry {

    enum RecordStates {
        NON_EXISTING,
        ACTIVE
    }

    // Data structure containing basic fields/attributes of a registry record.
    struct RecordAttributes {
        // Define record attributes here...
        uint z - ID;
    }

    // Data structure representing a single registry record.
    struct Record {
        // Record owner can be an external Ethereum account,
        // or a smart contract representing an user group or Organisation.
        address owner;

        RecordStates state;
        RecordAttributes attrs;
    }

    mapping(address => Record) records;
    address[] all_record_ids;

    // Admin of registry.
    address public registry_admin;

    // Registry constructor.
    function COMP6452 - studentsRegistry() public {
        // The user deploying the registry contract will be the registry admin.
        registry_admin = msg.sender;
    }

    // ====== CONTRACT EVENTS ======
}
```

Edit ERC20 Token Design

Does the token require details?

Details
 No Details

Name: COMP6452-Token
What the token will be called

Symbol: C6452

Decimals: 2

Is the token mintable?

Mintable
 Not Mintable

Is the token burnable?

Burnable
 Not Burnable

How will the token be initially distributed?

Account: e.g. 0x1434e8f21A0d8A66024E06a45637664b9349A691
Account to distribute allocation to

Allocation: 0
How much this account will be allocated

Add

[Compile Smart Contract](#)[Save & Close](#)[Cancel](#)

Smart Contract Output

[Source Code ▾](#)

```
pragma solidity ^0.4.24;

/**
 * @title SafeMath
 * @dev Math operations with safety checks that revert on error
 */
library SafeMath {

    /**
     * @dev Multiplies two numbers, reverts on overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns(uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b);

        return c;
    }

    /**
     * @dev Integer division of two numbers truncating the quotient, reverts on division by
     */
    function div(uint256 a, uint256 b) internal pure returns(uint256) {
        require(b > 0); // Solidity only automatically asserts when dividing by 0
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

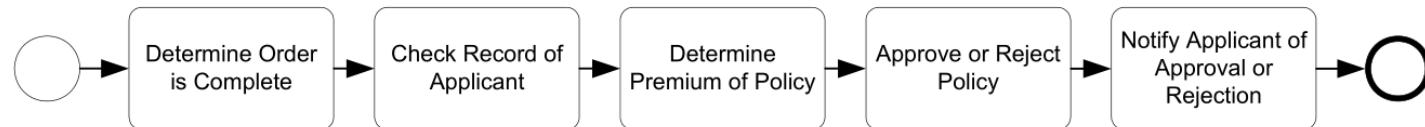
    /**
     * @dev Subtracts two numbers, reverts on overflow (i.e. if subtrahend is greater than min)
     */
    function sub(uint256 a, uint256 b) internal pure returns(uint256) {
        require(b <= a);
        uint256 c = a - b;
        return c;
    }
}
```



Business Process Model and Notation (BPMN)

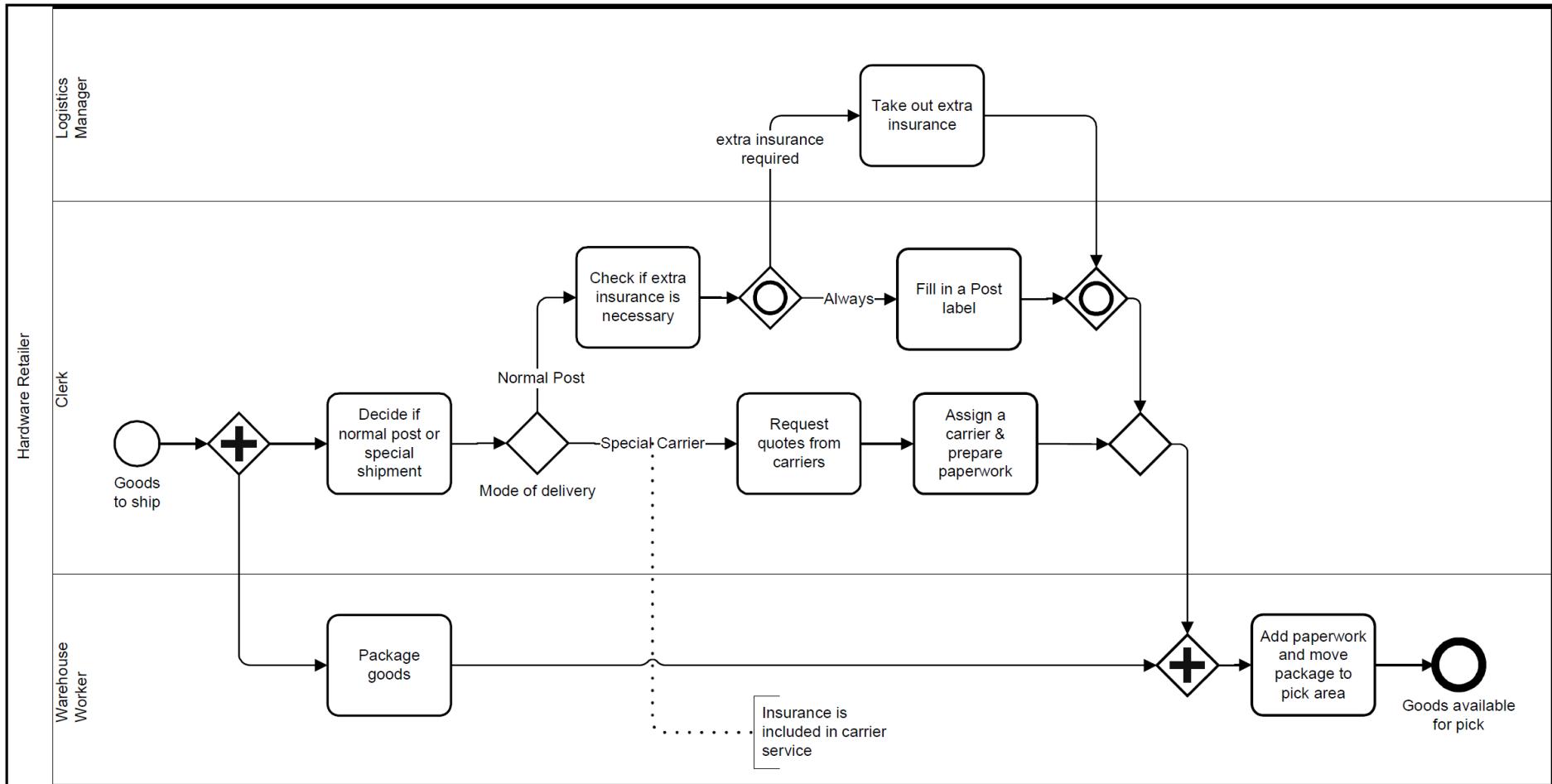
Business Processing Modelling: basic terminology

- BPMN example: insurance application processing



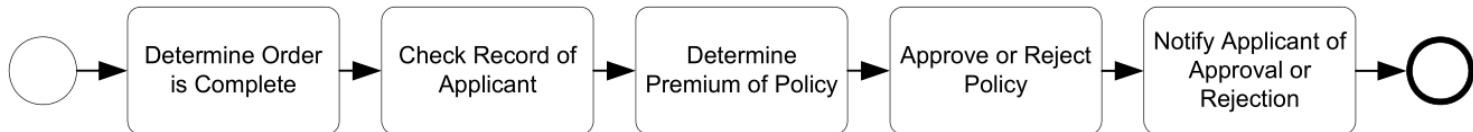
- Tasks: an activity that cannot be subdivided (unit of work)
- Sequence: some tasks need to be performed in a strict order
- Choice: certain tasks do not always need to be carried out
- Parallel: some tasks can be performed in parallel
- Synchronisation: some tasks need to wait for the result of previous tasks
- Iteration: some tasks need to be repeated

BPMN example: shipment process of a hardware retailer



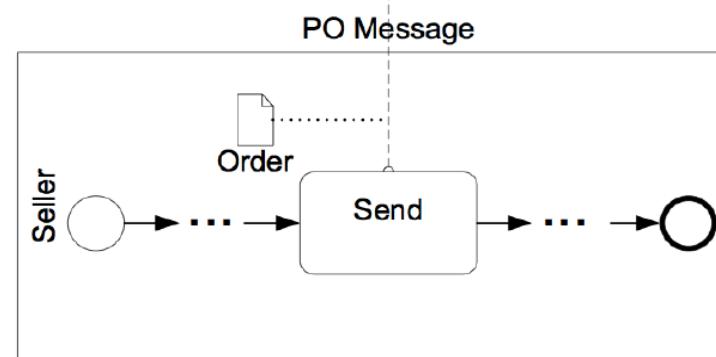
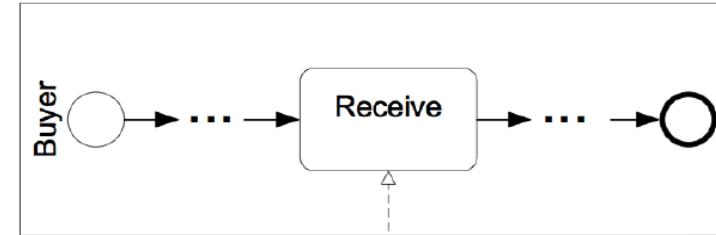
Business Process Model and Notation (BPMN): Motivation

- Focus of earlier workflow languages: clear execution semantics (e.g., PetriNets), expressivity (e.g., YAWL), interoperability (e.g., BPEL)
- Separate set of languages: conceptual process modelling notations without clear execution semantics, etc.
- Before BPMN, no notation existed that appealed to business users and has (somewhat) clear execution semantics
 - BPMN addresses this "niche"
- OMG Spec 2.0.2 in 2013: <http://www.omg.org/spec/BPMN/2.0.2/PDF>
 - From p.56 on, the elements are explained
 - (v1.0 released in 2006, v2.0 in 2011)
- BPMN example: private process

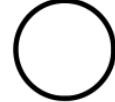
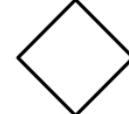


BPMN diagram elements

- Flow Objects
 - Events
 - Activities
 - Gateways
- Connecting Objects
 - Sequence Flow
 - Message Flow
 - Association
- Swimlanes
- Artefacts
 - Data Object
 - Group
 - Annotation



Core flow objects (from BPMN Spec)

Element	Description	Notation
Event	<p>An Event is something that “happens” during the course of a Process (see page 235) or a Choreography (see page 339). These Events affect the flow of the model and usually have a cause (<i>trigger</i>) or an impact (<i>result</i>). Events are circles with open centers to allow internal markers to differentiate different <i>triggers</i> or <i>results</i>. There are three types of Events, based on when they affect the flow: Start, Intermediate, and End.</p>	
Activity	<p>An Activity is a generic term for work that company performs (see page 149) in a Process. An Activity can be atomic or non-atomic (compound). The types of Activities that are a part of a Process Model are: Sub-Process and Task, which are rounded rectangles. Activities are used in both standard Processes and in Choreographies.</p>	
Gateway	<p>A Gateway is used to control the divergence and convergence of Sequence Flows in a Process (see page 147) and in a Choreography (see page 335). Thus, it will determine branching, forking, merging, and joining of paths. Internal markers will indicate the type of behavior control.</p>	

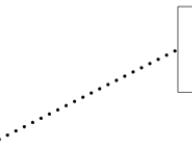
Connecting objects (from BPMN Spec)

Sequence Flow	A Sequence Flow is used to show the order that Activities will be performed in a Process (see page 95) and in a Choreography (see page 320).	
Message Flow	A Message Flow is used to show the flow of Messages between two <i>Participants</i> that are prepared to send and receive them (see page 113). In BPMN, two separate Pools in a Collaboration Diagram will represent the two <i>Participants</i> (e.g., PartnerEntities and/or PartnerRoles).	
Association	An Association is used to link information and Artifacts with BPMN graphical elements (see page 65). Text Annotations (see page 69) and other Artifacts (see page 64) can be Associated with the graphical elements. An arrowhead on the Association indicates a direction of flow (e.g., data), when appropriate.	

Swimlanes (from BPMN Spec)

Pool	<p>A Pool is the graphical representation of a <i>Participant</i> in a Collaboration (see page 113). It also acts as a “swimlane” and a graphical container for partitioning a set of Activities from other Pools, usually in the context of B2B situations. A Pool MAY have internal details, in the form of the Process that will be executed. Or a Pool MAY have no internal details, i.e., it can be a “black box.”</p>	<table border="1" data-bbox="1167 397 1666 561"> <tr> <td>Name</td><td></td></tr> </table>	Name	
Name				
Lane	<p>A Lane is a sub-partition within a Process, sometimes within a Pool, and will extend the entire length of the Process, either vertically or horizontally (see on page 304). Lanes are used to organize and categorize Activities.</p>	<table border="1" data-bbox="1167 697 1666 861"> <tr> <td>Name</td><td>Name</td></tr> </table>	Name	Name
Name	Name			

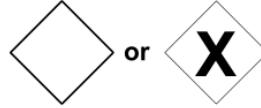
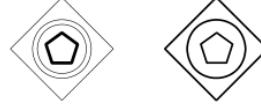
Artefacts (from BPMN Spec)

Data Object	<p>Data Objects provide information about what Activities require to be performed and/or what they produce (see page 204). Data Objects can represent a singular object or a collection of objects. Data Input and Data Output provide the same information for Processes.</p>	
Message	<p>A Message is used to depict the contents of a communication between two <i>Participants</i> (as defined by a business PartnerRole or a business PartnerEntity—see on page 91).</p>	
Group (a box around a group of objects within the same category)	<p>A Group is a grouping of graphical elements that are within the same Category (see page 68). This type of grouping does not affect the Sequence Flows within the Group. The Category name appears on the diagram as the group label. Categories can be used for documentation or analysis purposes. Groups are one way in which Categories of objects can be visually displayed on the diagram.</p>	
Text Annotation (attached with an Association)	<p>Text Annotations are a mechanism for a modeler to provide additional text information for the reader of a BPMN Diagram (see page 69).</p>	 <div style="border: 1px solid black; padding: 5px; display: inline-block;"> Descriptive Text Here </div>

Start, intermediate, end events (from BPMN Spec)

Start	As the name implies, the Start Event indicates where a particular Process (see page 235) or Choreography (see page 339) will start.	Start
Intermediate	Intermediate Events occur between a Start Event and an End Event. They will affect the flow of the Process (see page 237) or Choreography (see page 340), but will not start or (directly) terminate the Process.	Intermediate
End	As the name implies, the End Event indicates where a Process (see page 245) or Choreography (see page 343) will end.	End

Gateways (from BPMN Spec)

Gateway Control Types	<p>Icons within the diamond shape of the Gateway will indicate the type of flow control behavior. The types of control include:</p> <ul style="list-style-type: none"> Exclusive decision and merging. Both Exclusive (see page 286) and Event-Based (see page 296) perform exclusive decisions and merging Exclusive can be shown with or without the "X" marker. Event-Based and Parallel Event-based gateways can start a new instance of the Process. Inclusive Gateway decision and merging (see page 291). Complex Gateway -- complex conditions and situations (e.g., 3 out of 5; page 294). Parallel Gateway forking and joining (see page 292). <p>Each type of control affects both the incoming and outgoing flow.</p>	<p>Exclusive</p>  <p>or</p>  <p>Event-Based</p>   <p>Parallel Event-Based</p>  <p>Inclusive</p>  <p>Complex</p>  <p>Parallel</p> 
------------------------------	--	--



Student Task

- In the next 5 minutes, model the process of taking this class:
 - What are the activities?
 - In which order do they occur?
 - What are decision points in the process, what can happen?
 - Draw a model

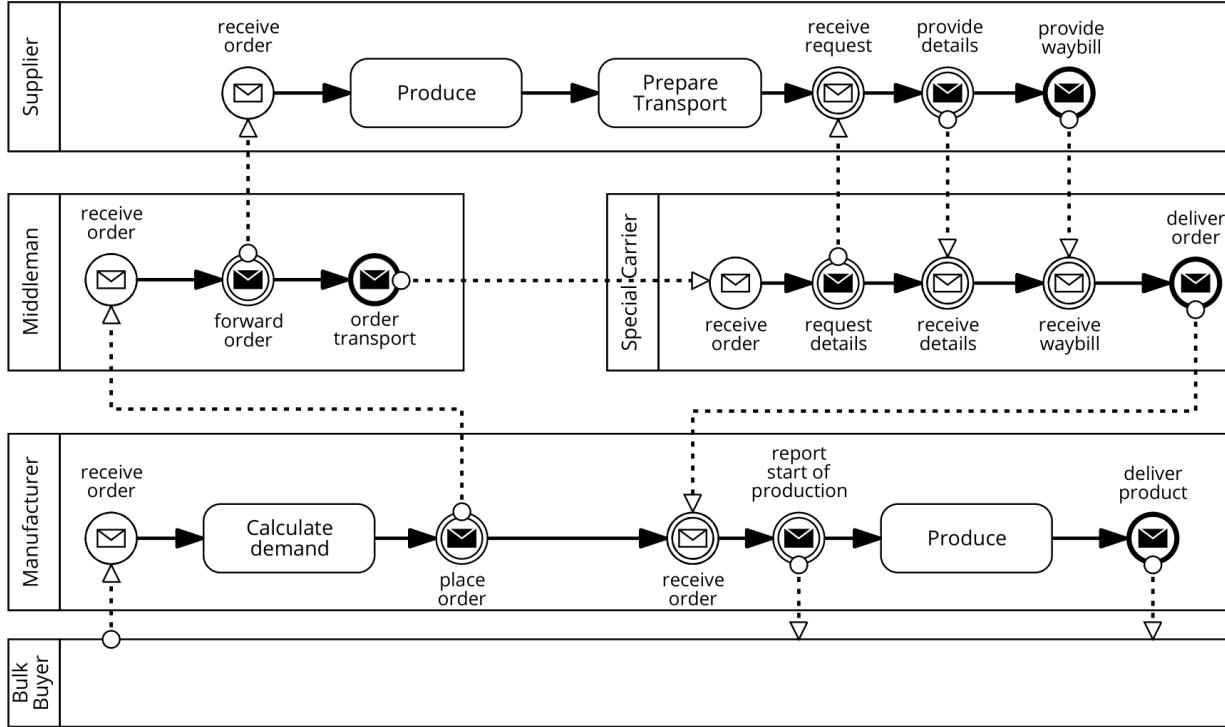


MDE for process models

MDE for Processes – Motivation

- Integration of business processes across organizations:
a key driver of productivity gains
- Collaborative process execution
 - Doable when there is trust – supply chains can be tightly integrated
 - Problematic when involved organizations have a **lack of trust** in each other
→ if 3+ parties should collaborate, where to execute the process that ties them together?
 - Common situation in “coopetition”

Motivation: example



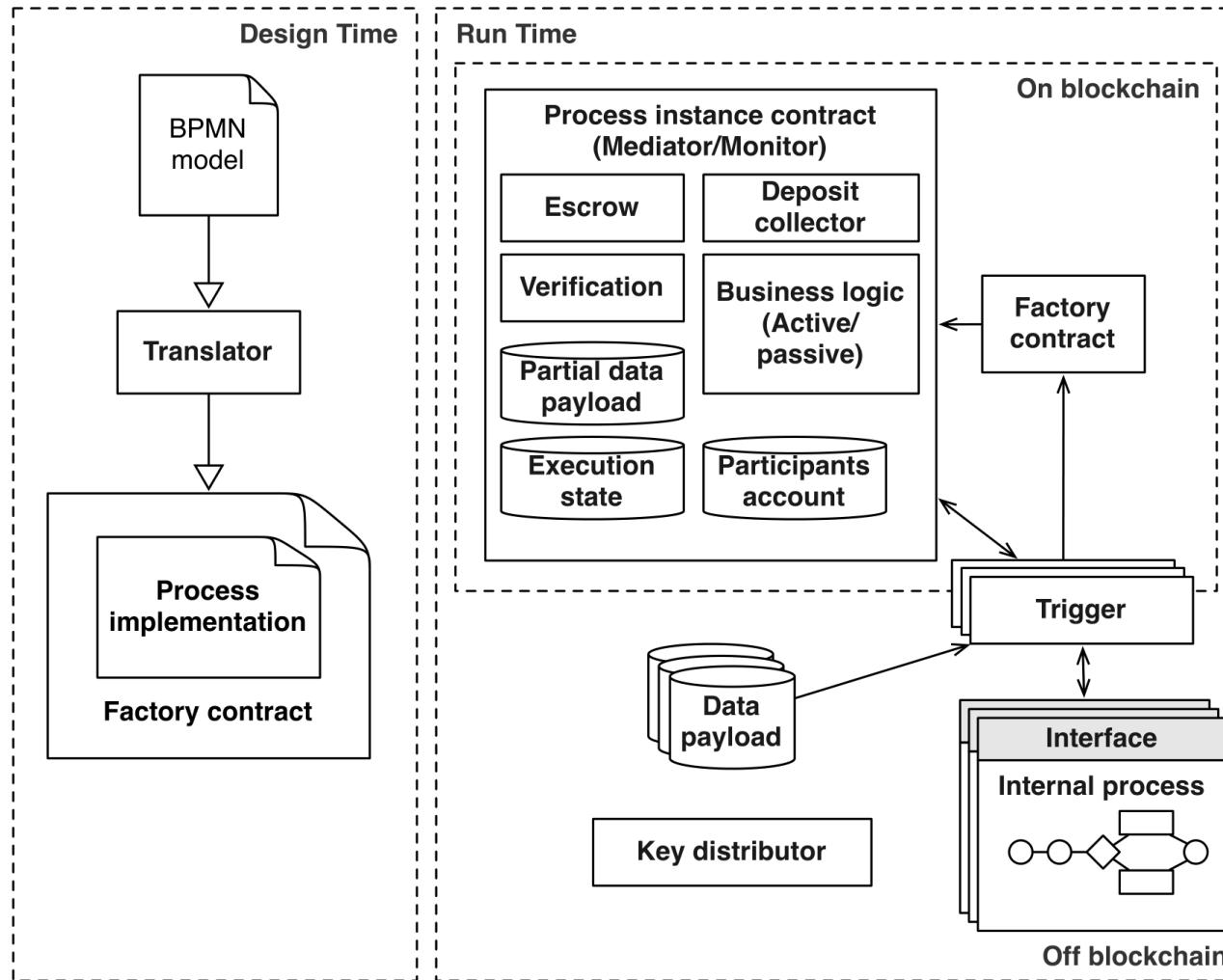
Issues:

- Knowing the status, tracking correct execution
- Handling payments
- Resolving conflicts

Approach in a nutshell

- Goal: implement collaborative business processes as smart contracts
 - Translate (enriched) BPMN to smart contract code
 - Triggers act as bridge between Enterprise world and blockchain
 - Smart contract provides:
 - Independent, global process monitoring
 - Conformance checking and process enforcement: only expected messages are accepted, only from the respective role
 - Automatic payments & escrow
 - Data transformation
 - Encryption
- Processes can make use of data / token contracts
 - Process activity to hand over title to a car / shipment / grain / ..., e.g., in exchange for fungible tokens

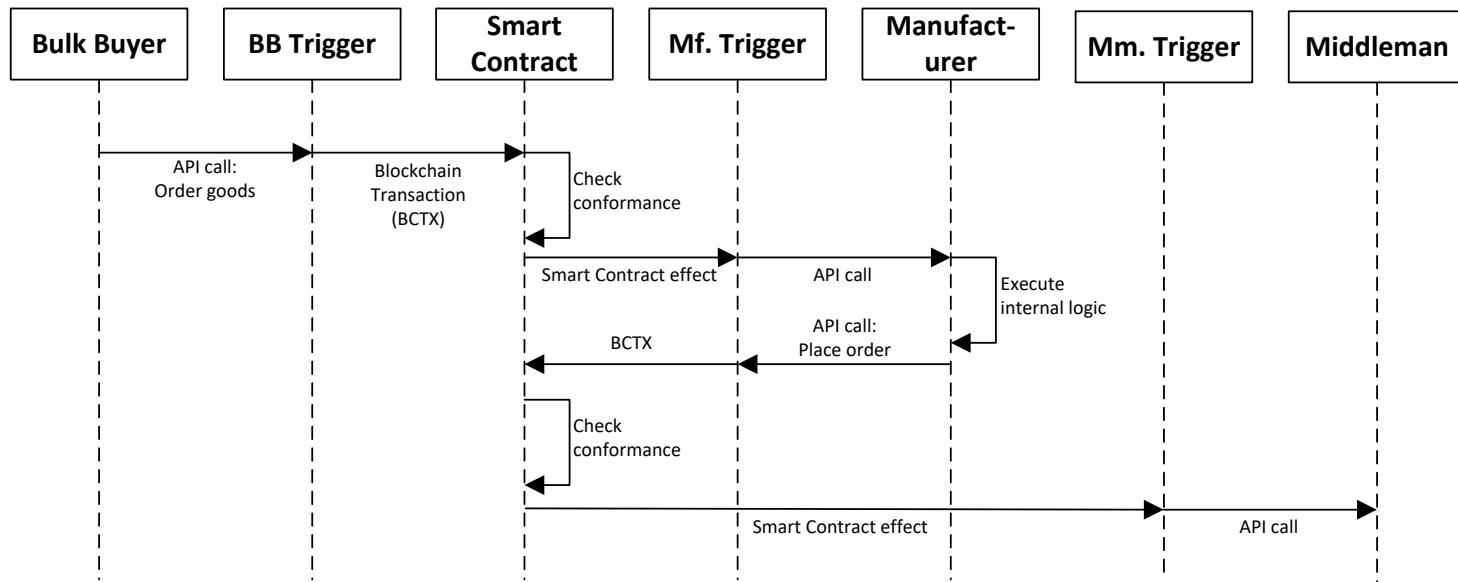
Architecture overview



Runtime

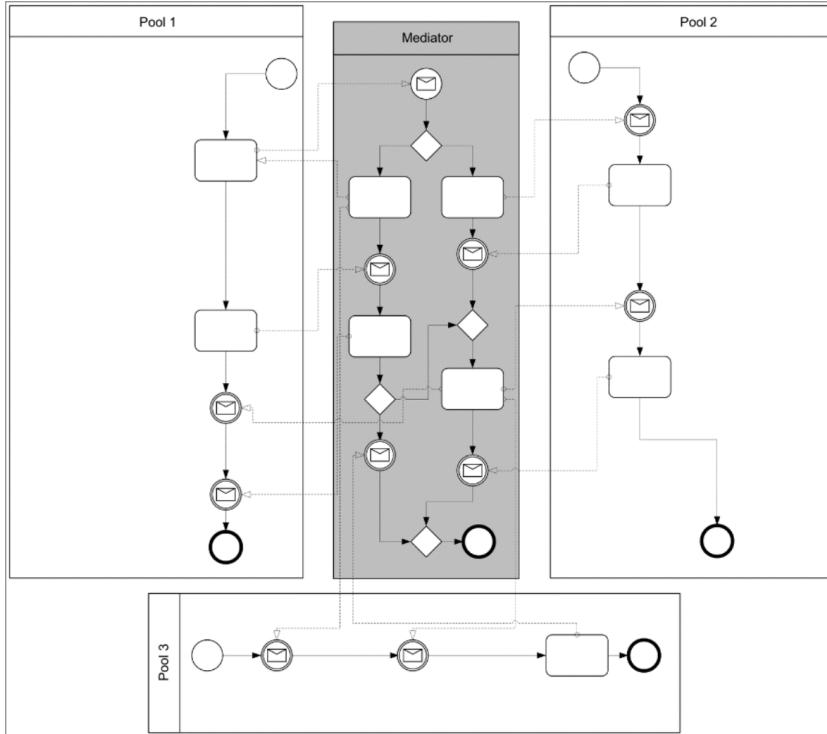
- Instantiation:
 - New *instance contract* per process instance
 - Assign accounts to roles during initialization
 - Exchange keys and create secret key for the instance
 - Messaging:
 - Instead of sending direct WS calls: send through triggers & smart contract
 - Instance contract handles:
 - Global monitoring
 - Conformance checking
 - Automated payments*
 - Data transformation*
- (*: optional, if certain assumptions are met)

Runtime – UML Sequence Diagram

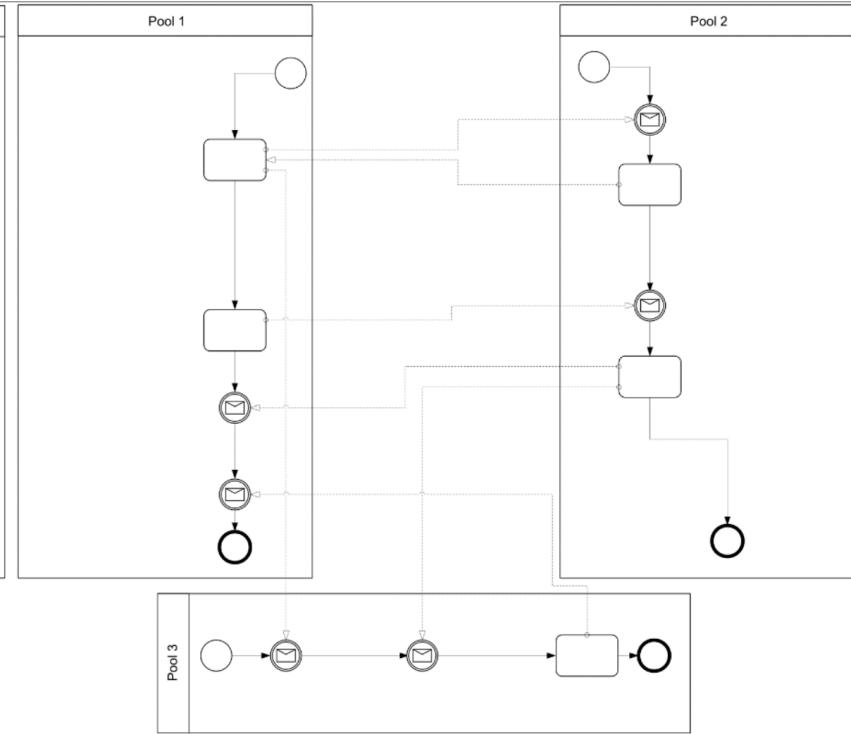


Collaborative processes: variants

Mediator (orchestration)

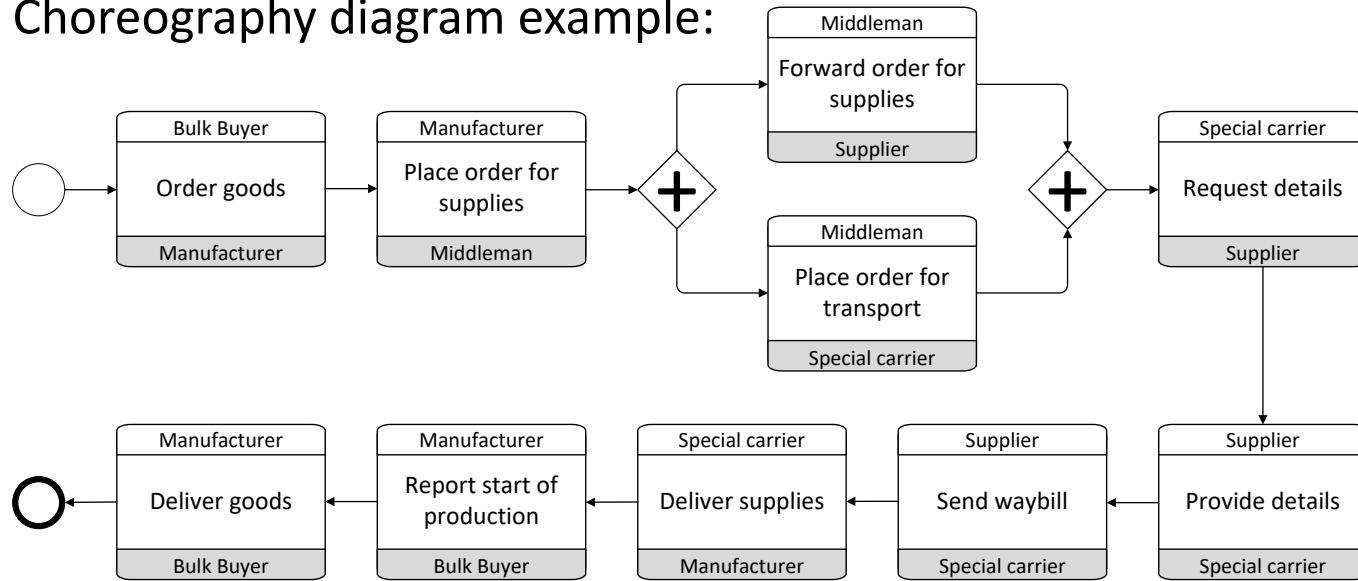


Choreography → C-Monitor

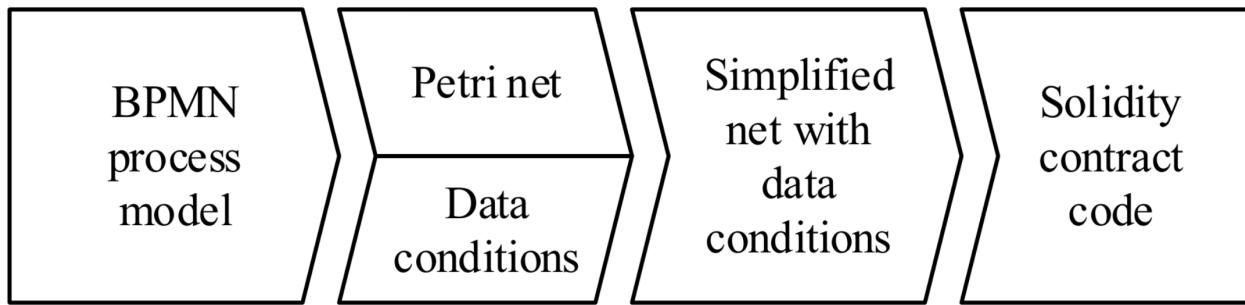


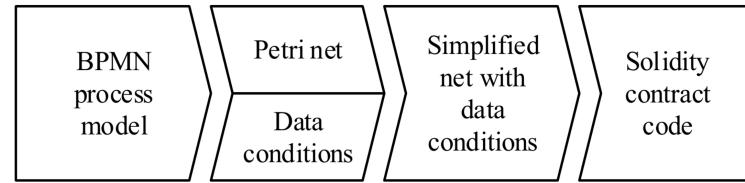
Translator

- Translate subset of BPMN elements to Solidity
 - BPMN Choreography diagrams or regular BPMN models with pools
 - BPMN Choreography diagram example:



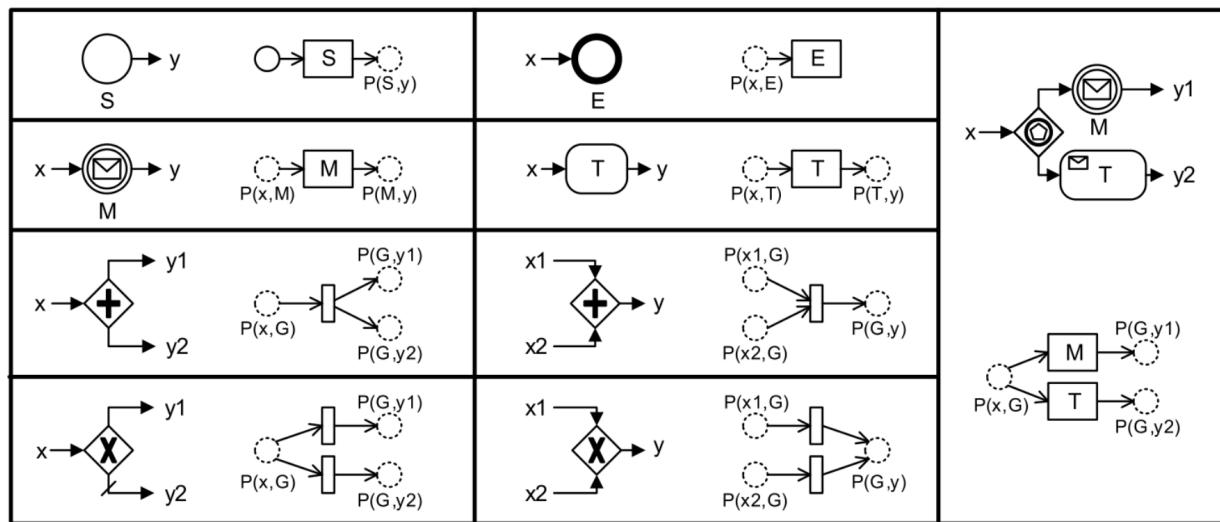
Translator

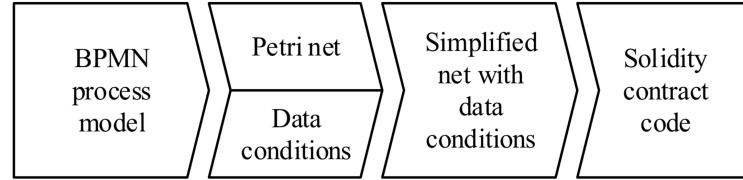




Translator

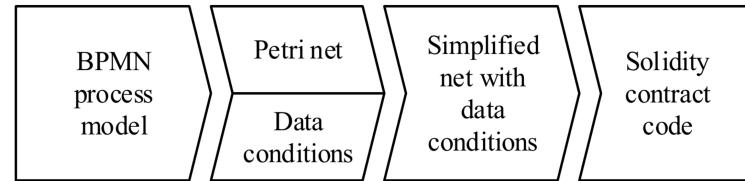
1. Translate BPMN control flow to WFnet (proven to be safe)





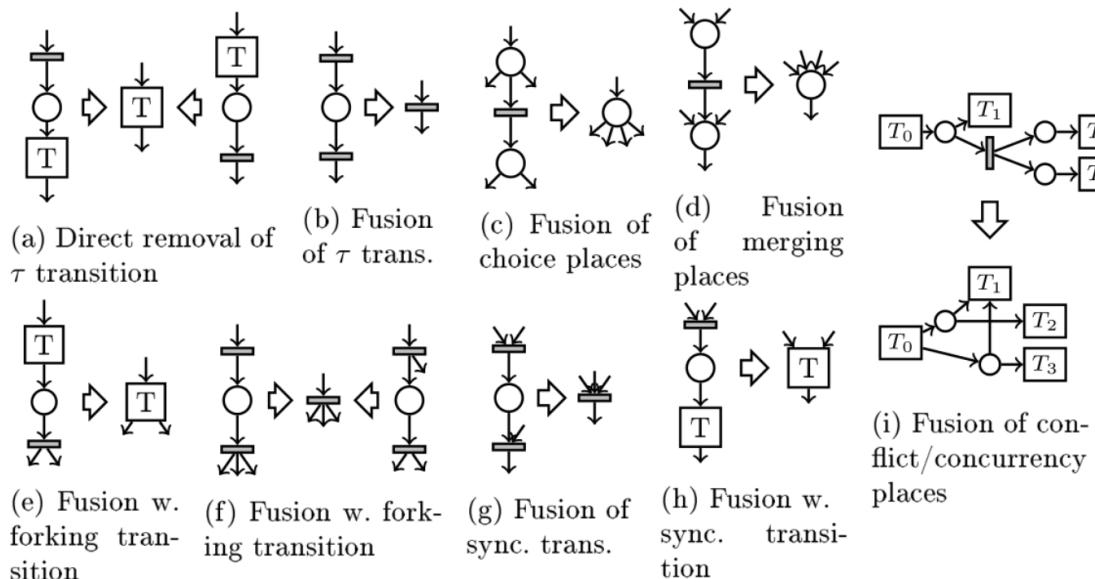
Translator

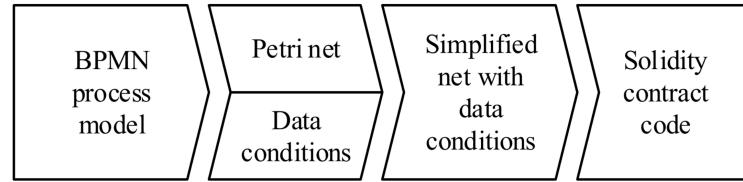
1. Translate BPMN control flow to WFnet (proven to be safe)
2. Capture dataflow and conditions



Translator

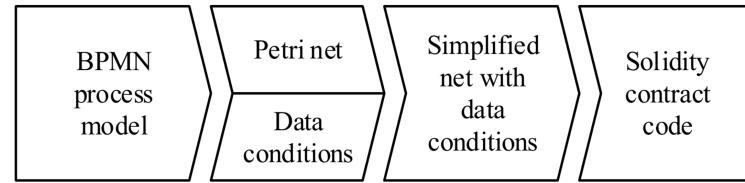
1. Translate BPMN control flow to WFnet (proven to be safe)
2. Capture dataflow and conditions
3. Reduce WFnet and annotate dataflow





Translator

1. Translate BPMN control flow to WFnet (proven to be safe)
2. Capture dataflow and conditions
3. Reduce WFnet and annotate dataflow
4. Translate into Solidity code
 - Status of the process is kept in a bit vector
 - Updates are bit-wise operations



Translator

1. Translate BPMN
2. Capture data
3. Reduce WFnet
4. Translate into Solidity
 - Status of the contract
 - Updates are bit vector operations

Bit vector check: does pos 1 have value “1”?

```

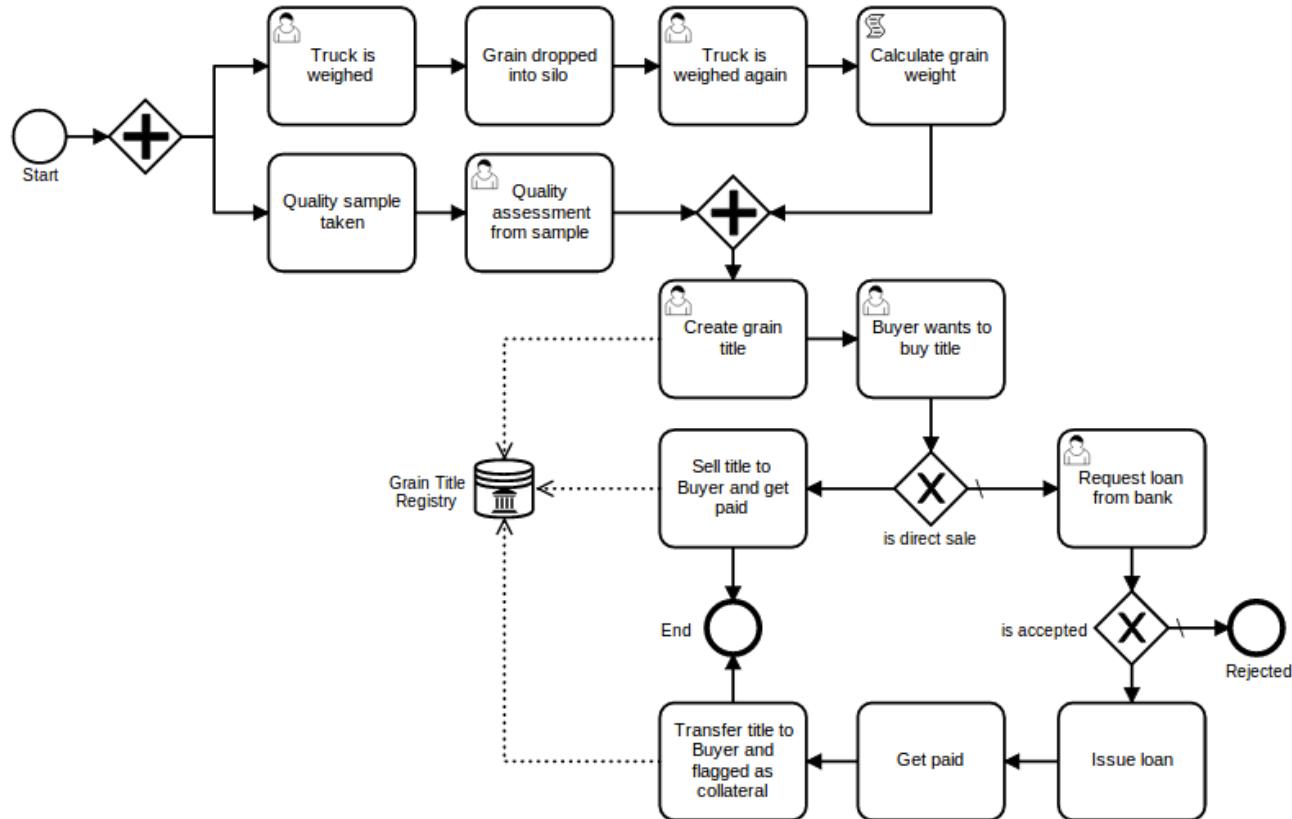
1 contract BPMNContract {
2   uint marking = 1;
3   uint predicates = 0;
4
5   function CheckApplication( – input params – ) returns (bool) {
6     if (marking & 2 == 2) { // is there a token in place p1?
7       // Task B’s script goes here, e.g. copy value of input params to contract variables
8       uint tmpPreds = 0;
9       if ( - eval P - ) tmpPreds |= 1; // is loan application complete?
10      if ( - eval Q - ) tmpPreds |= 2; // is the property pledged?
11      step (
12        marking & uint(~2) | 12, // New marking
13        predicates & uint(~3) | tmpPreds // New evaluation for “predicates”
14      );
15      return true;
16    }
17    return false;
18  }

```

Bit vector update: set pos 1 to “0”

set pos 2 and 3 to “1”

Combining process and data/token models





Tooling: Design time

Lorikeet Home Design Manage

Welcome 0x180d34b876DAa90057B2Ec345E82E2B1E9a4A082 Log Out

Edit BPMN Design

Business Process Name: GrainSupplyChain

Use Petri-Net Method:

Save Model Source XML

RegistryReference_03z5ksr

General Listeners Extensions Registry Reference

General

Id: RegistryReference_03z5ksr

Smart Contract Output

Solidity Code

```
// ----- REGISTRY INTERFACES
contract GrainTitleRegistry {
    function record_create(uint weight) returns(uint record_id);

    function record_transfer_ownership(uint record_id, address buyer_address, bool is_collateral);
}
// ----- 

contract ProcessMonitor {
    //uint preconditions = 0x800;

    function getPreconditions(uint instanceID) internal returns(uint);

    function setPreconditions(uint instanceID, uint preconditions) internal;
    event taskCompleted(uint indexed instanceID, string taskName);

    // ----- PROCESS VARIABLES
    bool isLoanAccepted;
    bool isDirectSale;
    bytes32 titleId;
    address buyerAddress;
    // ----- 

    // ----- REGISTRY CONTRACT ADDRESSES
    address addressOfGrainTitleRegistry = 0x11D6fd252049f869349CAdf4E2df3E17c8539Bf0;
    // ----- 

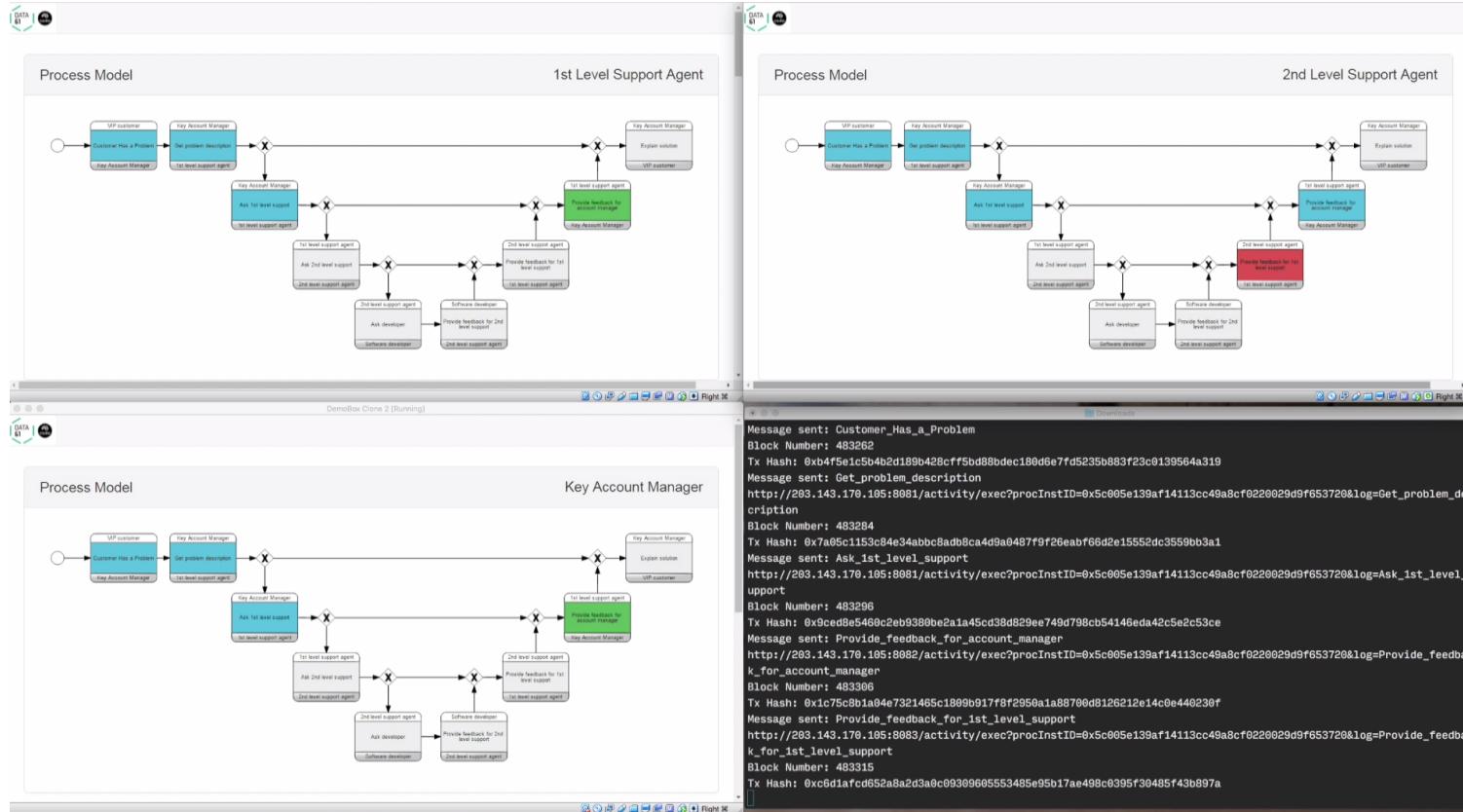
    function ProcessMonitor() {
        //
        //
        //
    }

    function Issue_loan(uint256 instanceID) returns(bool) {
        uint preconditions = getPreconditions(instanceID);

        if ((preconditions & (0x2 | 0x10000) == (0x2 | 0x10000)) {
            step(instanceID, preconditions & uint(~0x2) | 0x40);
            taskCompleted(instanceID, "Issue_loan");
            return true;
        }
    }
}
```

- Demo video: <https://drive.google.com/file/d/1rpy-oHbDVkXa6u4Fn73wSX8rlNn1sv3U/view>

Tooling: Runtime



MDE & related tools for blockchain

- Lorikeet (and its predecessor Regerator & BPM work)
 - Tools are not publicly available, but lots of information/papers on our website: <https://research.csiro.au/data61/blockchain/>
- Caterpillar – open-source BPMN execution engine (Uni Tartu & Data61)
 - <https://github.com/orlenyslp/Caterpillar>
- Regis – registry generator
 - <https://regis.nu/>
- Hyperledger Composer – for HLF business networks
 - <https://www.hyperledger.org/projects/composer>
- Other organisations (including Consensys) are working on tools, but most are not publicly available as yet

Summary

- Model-Driven Engineering basics
- MDE for data and token model
- Business Process Model and Notation (BPMN)
- MDE for process models



Thank You

Helen Paik

- | Senior Lecturer @ CSE, UNSW
- | Visiting Researcher @ AAP team, CSIRO Data61
- | h.paik@unsw.edu.au