课题编号： 2022YFB2702205 密级：公开

智能合约领域特定语言族(DSLs)

设计方案（初稿）

博雅正链（北京）科技有限公司

2024年4月

目 录

[1 智能合约领域特定语言族（DSLs）框架 1](#_Toc18682)

[1.1 需求分析 1](#_Toc26515)

[1.2 设计目标 2](#_Toc25035)

[1.2.1 业务流程的图形化开发 2](#_Toc22343)

[1.2.2 基于业务流程图生成智能合约 2](#_Toc22485)

[1.3 本项目支持的BPMN元素列表 3](#_Toc19157)

[1.4 BPMN模块介绍 5](#_Toc3272)

[1.4.1 总览 5](#_Toc19269)

[1.4.2 事件 5](#_Toc17216)

[1.4.3 活动 6](#_Toc16303)

[1.4.4 任务 6](#_Toc24879)

[1.4.5 网关 9](#_Toc11379)

[1.4.6 流对象 9](#_Toc7889)

[1.4.7 数据 10](#_Toc24809)

[1.5 BPMN基础样例展示 11](#_Toc19960)

[1.5.1 总体说明 11](#_Toc12277)

[1.5.2 数据流表示方式 11](#_Toc1261)

[1.5.3 控制流表示方式 13](#_Toc29160)

[2 金融领域DSL 14](#_Toc4419)

[2.1 需求分析 14](#_Toc17025)

[2.2 国内外相关研究工作 16](#_Toc31302)

[2.2.1 面向监管与合规的编程语言 16](#_Toc17444)

[2.2.2 智能合约编程语言中的规则数字化 18](#_Toc22518)

[2.3 设计目标 19](#_Toc14703)

[2.4 语义模型 19](#_Toc10915)

[2.5 程序结构 21](#_Toc4316)

[2.6 语法设计 23](#_Toc168)

[2.7 领域示范应用 29](#_Toc6672)

[2.7.1 领域场景 29](#_Toc25312)

[2.7.2 示范应用 29](#_Toc6920)

[3 政务领域DSL 31](#_Toc1470)

[3.1 需求分析 31](#_Toc27457)

[3.2 国内外相关研究工作 34](#_Toc27701)

[3.2.1 访问控制模型 34](#_Toc11663)

[3.2.2 面向流程管理的领域特定语言 35](#_Toc21203)

[3.3 设计目标 36](#_Toc17728)

[3.4 语义模型 36](#_Toc11451)

[3.5 语法设计 39](#_Toc18874)

[3.5.1 静态角色绑定语法设计 41](#_Toc2398)

[3.5.2 动态角色管理语法设计 41](#_Toc5054)

[3.5.3 语法/词法解析实例 42](#_Toc12587)

[3.6 领域示范应用 43](#_Toc6632)

[3.6.1 领域场景 43](#_Toc30862)

[3.6.2 示范应用 43](#_Toc15925)

[4 民生领域DSL 45](#_Toc1001)

[4.1 需求分析 45](#_Toc31876)

[4.2 国内外相关研究工作 47](#_Toc19577)

[4.3 语义模型 49](#_Toc10875)

[4.4 程序结构 51](#_Toc30018)

[4.5 语法设计 53](#_Toc15157)

[4.5.1 签名语法设计 56](#_Toc14454)

[4.5.2 承诺语法设计 57](#_Toc19864)

[4.5.3 零知识证明语法设计 58](#_Toc14578)

[4.6 领域示范应用 59](#_Toc16980)

[4.6.1 领域场景 59](#_Toc8391)

[4.6.2 示范应用 59](#_Toc16085)

[5 技术实现 61](#_Toc31978)

[5.1 系统总体设计 61](#_Toc27811)

[5.1.1 系统功能模块概览 61](#_Toc1916)

[5.1.2 BPMN组件与DSL组件的关系 61](#_Toc277)

[5.1.3 前端BPMN业务流程建模模块 62](#_Toc23888)

[5.1.4 后端语义转换模块 63](#_Toc14140)

[5.1.5 后端生成的智能合约 64](#_Toc16531)

[5.2 前端技术选型 64](#_Toc30312)

[5.2.1 Vue 65](#_Toc22982)

[5.2.2 ElementUI-Plus 65](#_Toc29127)

[5.2.3 BPMN-JS 65](#_Toc12644)

[5.2.4 Vuex 66](#_Toc12766)

[5.2.5 Axios 66](#_Toc12271)

[5.2.6 Vite 66](#_Toc16112)

[5.3 前端关键技术实现 67](#_Toc7997)

[5.3.1 添加自定义BPMN模块 67](#_Toc7065)

[5.3.2 下载流程图 69](#_Toc16175)

[5.3.3 读取流程图 71](#_Toc31001)

[5.3.4 链下模拟运行 71](#_Toc20809)

[5.3.5 链上运行 73](#_Toc6950)

[5.4 后端关键技术实现 75](#_Toc22963)

[5.4.1 概览 75](#_Toc20299)

[5.4.2 系统后端设计概述 76](#_Toc2486)

[5.4.3 生成的合约框架 77](#_Toc15374)

[5.4.4 控制流转换过程 79](#_Toc9906)

[5.4.5 BPMN数据流转换过程 84](#_Toc11891)

[6 参考文献 88](#_Toc6678)

# 智能合约领域特定语言族（DSLs）框架

## 需求分析

智能合约是一类在区块链上编写、实现、执行的程序。相比于传统链下执行的程序，智能合约具备了区块链系统的防伪造和不可篡改等特性；同时，智能合约的运行结果存储在分布式的区块链账本中，保证了运行结果的透明性和唯一性。智能合约基于区块链执行的特点令其成为金融、政务、民生等创新应用场景的良好方案。然而据调查结果显示，当前阻碍智能合约更广泛应用的因素在于其编写难度大，掌握以Solidity为代表的智能合约编程语言的人较少，使得领域专家无法直接参与链上智能合约的设计与编写。从而使得领域专家无法将领域内的业务需求有效转化为可解决需求的智能合约，阻碍了智能合约技术在这些特定领域内的实际应用，领域专家只得采取传统链下执行的方式实现需求。为此，本课题设计了基于BPMN[[[1]](#endnote-0)]的业务流程链上执行引擎，并嵌入三种领域特定语言模块，使得领域专家可以通过低代码的方式直接参与业务流程建模，完成通用业务流程的设计与领域特定业务流程的编写。本章将针对通用业务流程模型进行需求分析。领域特定的业务流程编程语言的需求分析将在其各自章节中进行介绍。

在目前基于区块链的业务场景中，针对业务场景进行建模的需求主要包括粗粒度的BPMN业务流程框架设计和细粒度的BPMN元素详细设计。前者包括领域专家使用BPMN元素符号对业务流程进行模拟；后者包括领域专家提供每个BPMN符号转换为智能合约代码的具体细节（以低代码形式实现）。因此，一个典型的业务场景的流程建模与执行主要包括以下几个步骤：

1） 领域专家、业务人员可以通过开发界面拖拽BPMN符号，搭建业务流程的总体框架，明确业务的功能划分、总体流程，并对流程的逻辑进行检验；

2） 领域专家、业务人员在BPMN模型基础上，对每一个具体的活动节点通过下拉框、输入框等低代码方式进行详细设计，对业务的数据流通过图形化的方式进行规定，对服务调用的具体信息，如接口、变量、消息等进行标识；

3） 将扩展后的业务模型转换为智能合约，部署到链上执行引擎上。引擎可以解析BPMN总体流程、数据流、符号内细节标注等所有内容，完成智能合约在区块链上的部署，并自动执行业务功能。

## 设计目标

具体的，本课题就基于BPMN的智能合约领域特定语言族开展研究，拟实现如下两个核心设计目标。

### 业务流程的图形化开发

业务流程的图形化开发用于提供可视化的开发界面，支持包括金融、政务、民生领域在内的领域专家进行业务流程的建模和设计。其可以分为以下子功能点：

1. 业务流模型定义子功能：从领域专家的角度提供BPMN建模标注子集元素，支持简单事件、任务、网关和顺序流等基本BPMN元素的建模，以及对复杂的外部用户任务、服务任务的支持。用户应使用上述流程定义工具进行流程模型的定义，基于BPMN规范使用XML标签形式来定义业务流程；
2. 业务流模型转换子功能：业务流模型是业务流程引擎﻿的基础和核心，一旦将不合理的业务流模型投入实际运营，将会造成无法弥补的损失。根据BPMN业务流模型的转换规则，生成与业务流模型相对应的XML文件，并对XML进行解析从而生成可序列化的JSON文件，用于前后端交互；
3. 领域特定语言扩展子功能：提供BPMN元素的扩展属性，为领域专家提供BPMN基本元素扩展的支持；提供增加自定义BPMN元素的功能，为领域专家提供实现自定义领域特定语言模块的支持；

### 基于业务流程图生成智能合约

基于业务流程图生成智能合约，是指系统后端在接收到来自前端（业务流程可视化开发界面）序列化的JSON文件后，根据BPMN元素规则，业务流程控制流，数据流等要素，生成与业务流程对应的智能合约。其可以分为以下子功能点：

1. BPMN基本元素转换子功能：根据BPMN基本元素的逻辑转换生成智能合约，支持简单事件、任务、网关、顺序流生成智能合约状态机逻辑，通过状态机驱动各个函数的先后运行关系，使其按照流程图的顺序调用执行；
2. 领域特定语言扩展元素转换子功能：根据本课题设计的领域特定语言的语法规则和语义模型，生成与用户输入的领域特定语言相对应的智能合约。需要将领域特定语言的转换规则嵌入到BPMN转换系统中；
3. 业务数据流转换子功能：根据业务数据在BPMN流程图中的顺序和执行关系，在智能合约中生成对应的数据传递与调用。其主要实现方式是，需要生成智能合约中函数的输入参数和返回值，以满足数据流传递的需求；

## 本项目支持的BPMN元素列表

BPMN开发的一个主要目标是创造出一个简单易懂的机制和规则来创建业务流程模型，同时也能够应对业务流程固有的复杂性。要处理这两个相互冲突的需求，所采取的方法是将标注的图标按照特定的类别进行组织。下文提供了一些本项目支持的表示BPMN流程图中基本类型的图标。在元素基本类别的基础上，业务用户可以添加额外的变化和信息以支持复杂性的需求，同时不会显著改变BPMN图的总体观感。

表 1 本项目支持的BPMN元素列表

|  |  |  |
| --- | --- | --- |
| 名称 | 说明 | 图标 |
| 开始事件 | ﻿开始事件表示一个特定流程的开始 |  |
| 结束事件 | ﻿结束事件表示一个流程的结束 |  |
| 中间事件 | ﻿中间事件发生在开始事件和结束事件之间。﻿其会通过抛出事件影响流程的运行，﻿但不会启动或终止流程 |  |
| 任务 | ﻿任务是包含在流程中的原子活动 |  |
| 用户任务 | ﻿﻿用户任务是需要外部参与者链下参与到流程中的任务 |  |
| 脚本任务 | ﻿脚本任务是由业务流程引擎执行的任务 |  |
| 发送任务 | ﻿发送任务是用于向外部参与者发送消息的任务 |  |
| 服务任务 | ﻿服务任务是使用某种外部服务的任务，如网络、数据库服务等 |  |
| 排他网关 | ﻿排他网关用于在流程中创建多条可选路径，但只能执行其中一条路径 |  |
| 并行网关 | ﻿并行网关会创建多条可同时执行路径，不检查任何约束条件 |  |
| 顺序流 | 顺序流用于表明流程中活动的执行顺序 |  |
| 条件顺序流 | 条件顺序流根据表达式的运行结果以确定是否使用顺序流 |  |
| 默认顺序流 | 默认顺序流﻿在所有条件顺序流运行不为真时执行 |  |
| 数据对象 | 数据对象表明业务流程中需要数据或生成的数据 |  |
| 数据对象集合 | ﻿数据对象集合是数据对象的扩展类型，表示多个数据元素的集合 |  |
| 数据存储 | ﻿数据存储用于存储流程范围之外的数据信息，并提供检索和更新 |  |
| 金融DSL任务 | 金融DSL任务是用户编写金融领域特定语言的任务，主要用于监管和审计金融交易。 |  |
| 政务DSL任务 | 政务DSL任务是用户编写政务领域特定语言的任务，主要用于权限控制和基于角色的自动化公务流程。 |  |
| 民生DSL任务 | 民生DSL任务是用户编写民生领域特定语言的任务，主要对常用民生服务使用密码学算法进行加密。 |  |

## BPMN模块介绍

### 总览

本项目支持（可以转化生成Solidity）的BPMN元素的基本类别有：

1. 流对象（Flow objects）

2. 数据（Data）

3. 连接对象（Connecting objects）

流对象是BPMN图中主要的元素，用于定义业务流程的行为。有以下三种流对象：活动（Activities）；事件（Events）；网关（Gateways）

数据包括以下两种对象：数据对象（Data objects）；数据存储（Data stores）

连接对象用于将流对象、数据之间相互连接，有以下三种连接对象：顺序流（Sequence flow）；条件顺序流（Condition flow）；默认顺序流（Default flow）。

### 事件

﻿事件是在流程过程中“发生”的事情。这些事件通常会影响流程的进行，并且通常具有相应的原因（触发条件）或影响（结果）。事件的BPMN图标是具有开放中心的圆圈，可以通过根据内部标识来区分不同的触发条件或结果。事件根据其影响流程的时机可分为三种：开始、中间、结束。

1. **开始事件**

开始事件表示一个流程的开始。因此不会有任何传入的顺序流，即任何流都不能连接到开始事件。开始事件的基本形状与中间事件和结束事件相似，即一个轮廓为细线的、中心开放的圆圈，可以在圆圈内放置标识以扩展事件表示。

1. **结束事件**

结束事件表示一个流程的结束。因此不会有任何传出的顺序流，即任何流都不能从结束事件出发。结束事件的基本形状与开始事件和中间事件相似，即一个轮廓为粗线的、中心开放的圆圈，可以在圆圈内放置标识以扩展事件表示。例如在圆圈内可以放置一个实心信封符号，代表可以在流程结束时抛出事件。



图 1 BPMN结束事件示意图



图 2 BPMN消息结束事件示意图

1. **中间事件**

中间事件表示流程开始和结束之间发生的事情。中间事件会影响流程的运行，但不会启动或终止流程。中间事件可用于：

* ﻿显示流程中期望或触发的消息；
* 显示流程中预期的延迟；
* 通过触发异常处理来中断正常流程；
* 显示所需的额外工作。

中间事件的基本形状与开始事件和结束事件相似，是一个中心开放的圆圈，轮廓为双细线，可以在圆圈内放置标识以扩展事件表示。

### 活动

活动是业务流程设计者在流程中所执行工作的通用术语。活动可能是原子性的，如任务；也可能是非原子性（复合性）的，如子流程。子流程和任务都是圆角矩形。在本项目中，支持针对原子性活动（任务）生成对应Solidity代码。

### 任务

﻿任务是包含流程在中的原子活动。当流程中的工作不能被细分为更精细的流程时，则使用任务，反之则使用子流程。

1. **服务任务**

服务任务是采用某种服务的任务，可以是网络服务、数据库服务、或是自动化应用。服务任务对象与任务具有相似的形状，然而其形状的左上角有一个齿轮标识，表示该任务是服务任务。服务任务为一个圆角矩形，必须由单条细线绘制，并在左上角包含一个齿轮标识将其与其他任务类型区分开来。

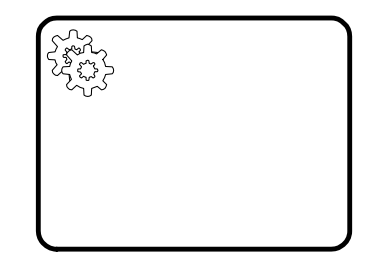


图 3 BPMN服务任务示意图

1. **用户任务**

用户任务是工作流中最典型的一类任务，需要由人工执行者在外部软件接口的帮助下执行任务。用户任务为一个圆角矩形，并在左上角包含一个人物标识，将用户任务与其他任务类型区分开来。

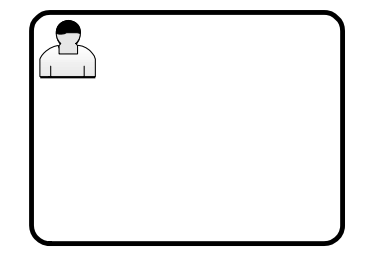


图 4 BPMN用户任务示意图

1. **发送任务**

发送任务旨在向外部参与者（相对于流程）发送消息。发送消息后，任务即完成。通过使用流程的定义协作中的消息流，将发送任务连接到参与者。发送任务与其他任务具有相同的形状，即圆角矩形。其形状的左上角有一个信封标识（与抛出消息事件的标识相同），表示该任务是发送任务。

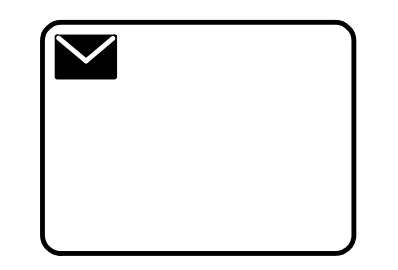


图 5 BPMN发送任务示意图

1. **脚本任务**

脚本任务由业务流程引擎执行。流程建模者用引擎可以解释的的语言定义脚本。当任务准备开始时，流程引擎将执行脚本；当脚本完成时，任务也标志着完成。脚本任务为一个圆角矩形，其形状左上角有一个脚本标识，表示该任务是脚本任务。

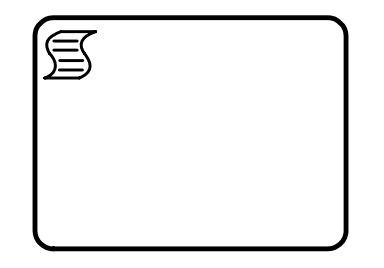


图 6 BPMN脚本任务示意图

1. **金融DSL任务**

金融DSL任务是为金融领域特定语言专门设置的一种任务，其用途是将金融领域特定语言转换为智能合约代码。本项目设计的金融领域特定语言RegLang，其用途是为了在区块链上编写基于规则的监管合约。为了便于监管专家将监管规则数字化，RegLang在保证对监管规则具有充分的表达能力的基础上，被设计得尽可能简单，并在类型系统、内置函数、逻辑量词的支持方式等方面提高数字化规则编写的便利性。金融DSL任务可以作为整体业务流程中的一部分，起到在业务流程中监管金融交易等作用。



图 7 BPMN金融DSL任务示意图

1. **政务DSL任务**

政务DSL任务是为政务领域特定语言专门设置的一种任务，其用途是将政务领域特定语言转换为智能合约代码。本项目设计的政务领域特定语言AuthLang，其用途是为了实现针对政务领域复杂访问控制逻辑的灵活管理与准确表达。AuthLang可以基于插件形式继承于BPMN框架中，以将BPMN流程编排为由不同角色参与方所实现的子流程。AuthLang也可以以独立政务DSL任务呈现，作为整体业务流程中的一部分，起到权限控制和基于角色的自动化公务流程的作用。



图 8 BPMN政务DSL示意图

1. **民生DSL任务**

民生DSL任务是为民生领域特定语言专门设置的一种任务，其用途是将民生领域特定语言转换为智能合约代码。本项目设计的民生领域特定语言CryptLang，其用途是实现针对民生领域复杂密码学逻辑的灵活管理与准确表达。CryptLang将基于插件形式继承于BPMN框架中，以将BPMN流程编排为由不同角色参与方所实现的子流程。功能性上，CryptLang将实现隐私数据的的加密与验证工作，实现复杂流程执行过程中的高效密码学API调用，提高开发效率。



图 9 BPMN民生DSL任务示意图

### 网关

网关用于控制流程中顺序流的发散和汇聚。“网关”一词意味着存在一种逻辑门机制，控制代表流程运行状态的令牌是否允许通过网关。网关为菱形，必须由单条细线绘制，根据菱形内部的标识区分其是排他、并行、或其它类型的网关。

1. **排他网关**

排他网关用于在流程中创建多条可选路径，其一般连接着多条输出的顺序流。对于流程而言，排他网关可以视为“道路上的分流点”，只能采用其中一条路径继续执行流程。排他网关可以被认为是在流程的特定点上提出的表达式，有一组明确的备选答案。每个答案都与一个条件表达式相关联，而表达式又与网关的传出顺序流相关联。

﻿建模者可以规定一条默认路径，以便在所有条件表达式的计算结果均不为真时，采用该路径。如果未指定默认路径，并且在执行流程时，所有条件表达式的计算结果均不为真，则会发生运行时异常。

1. **并行网关**

﻿并行网关用于同步并行流或创建并行流。﻿并行网关会创建多条并行输出路径，不检查任何条件并允许表示流程运行的令牌顺着所有传出流执行。对于传入流，并行网关将等待所有传入流完成输入，然后再同时触发其所有传出流。

### 流对象

﻿流用于表示各个活动在流程中的执行顺序，其连接图中的各个任务、活动、事件、数据等基本元素。

1. **顺序流**

﻿顺序流是流对象的基本类型，用一个基本的单箭头表示。

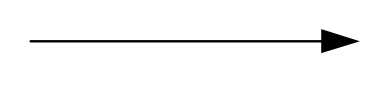


图 10 BPMN顺序流对象示意图

1. **条件顺序流**

顺序流可能会具有一个条件表达式，在运行时进行评估以确定是否将使用该顺序流。如果条件顺序流是从活动传出的，那么该顺序流中，在连线的开头处会有一个小菱形。如果条件顺序流是从网关传出的，那么线条上则没有小菱形。

图 11 BPMN条件顺序流对象示意图

1. **默认****顺序流**

对于基于表达式的排他网关，默认顺序流是只有当所有其他的传出条件顺序流在运行时都不为真时，才会执行的流。对于默认顺序流，在连线的开头处会有一个斜杠。



图 12 BPMN默认数据流对象示意图

### 数据

1. **数据对象**

﻿数据对象表示活动在执行时所需要的或产出的信息，其可以代表单个对象或对象集合。数据输入和数据输出为各个流程提供相同的信息。

属性isCollection定义了数据对象是否表示的是元素集合，如果表示的是元素集合，则该集合由多个数据对象组成。



图 13 BPMN单个数据对象示意图



图 14 BPMN数据对象集合示意图

1. **数据存储**

数据存储提供了一种可以让活动对存留在流程范围之外的信息进行检索或更新的机制。其代表将流程图中的数据存储起来，之后可以在流程图中多次引用，使不同的活动或任务都可以获取到数据存储中相同的数据。

## BPMN基础样例展示

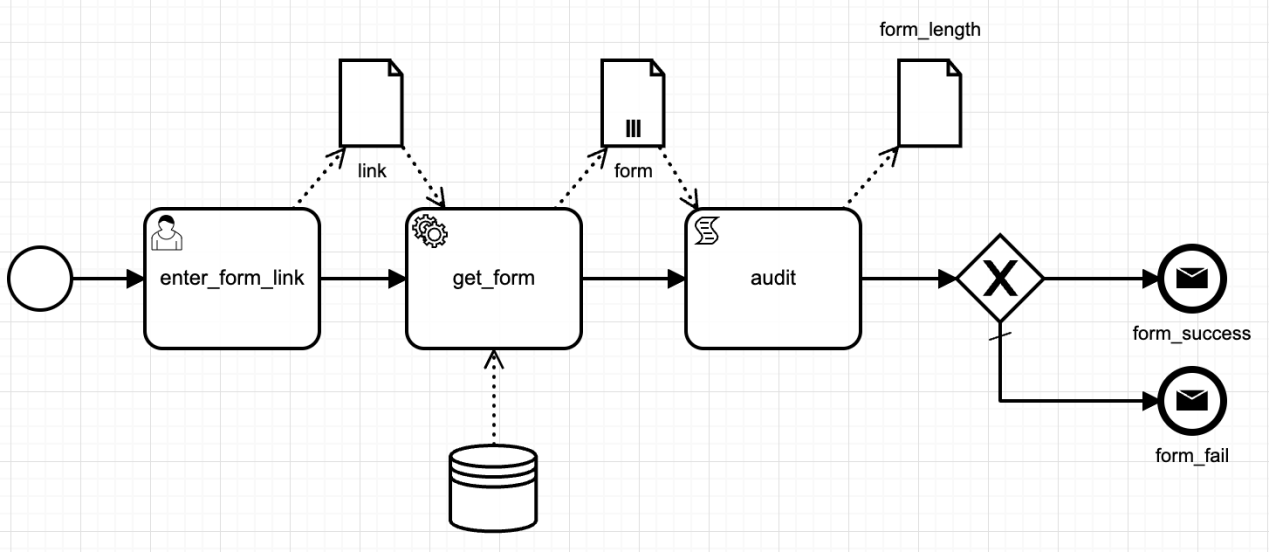


图 15 BPMN基础样例示意图

### 总体说明

该样例流程的目标是对用户提交的表单进行审计，判断表单的文件长度是否符合条件并以事件的形式返回结果。与传统流程管理平台不同的是，本项目使用链上平台管理流程的执行，充分利用区块链可审计、防篡改的优势。该样例流程涉及到了本项目所支持的大部分BPMN元素，包括用户任务（输入表单地址）、服务任务（向外部数据源请求数据）、脚本任务（执行计算文件长度的脚本），排他门（判断长度是否符合要求），消息结束事件。

### 数据流表示方式

如下图所示，对于外部任务（包括用户任务、服务任务等），若数据对象通过虚线指向该任务，则代表该数据对象是已存储在流程全局区的一个变量，且需要由该外部任务提供给外部对象的数据；若数据对象是被一条从外部任务出发的虚线所指，则代表该数据对象由该外部任务提供，并存储在流程的全局区中。如样例所示，经过用户任务enter\_form\_link之后，得到用户输入的表单地址link并存储在流程全局区中。随后服务任务get\_form根据已存储在全局区的变量link的值，通过数据库服务在数据库中获取该link对应的表单，并将表单存储在流程全局区中。注意到，由虚线指向该服务任务的是数据对象link，代表link是服务任务get\_form提供给外部服务的数据；从该服务任务由虚线指出的数据对象是form，代表form是由服务任务get\_form从外部得到的数据。类似的，对于脚本任务audit，其执行过程不涉及外部对象，仅在流程引擎内部执行脚本。因此该脚本任务的数据流是以表单数据对象form作为输入，计算并输出数据对象form\_length，其代表表单文件长度，同时也是后续排他门的判断标准。

每一个在图中定义BPMN数据对象元素都会转换为Solidity中的变量对象，因此在绘制BPMN流程图时，需要对数据对象的数据类型进行标注。如下图所示，本项目支持Solidity中常用的几种数据类型：有符号整型、无符号整型、字符串类型、字符数组类型、布尔类型、数组类型。

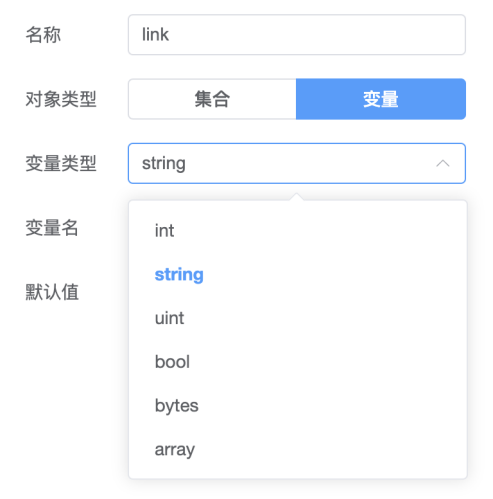


图 16 BPMN支持的数据类型集合

若该数据对象类型为集合，则用户可以在集合中添加指定类型的变量。BPMN中的数据对象集合在生成的Solidity中为结构体类型。下图即本流程中的数据对象集合form（用户提交的表单）的一个样例，包含了字符串类型的姓名、简历，和无符号整型的年龄字段。在BPMN流程图中，form也以数据对象集合的形式进行了标注。

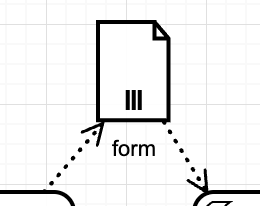


图 17 BPMN中添加变量方式

### 控制流表示方式

BPMN流程的执行逻辑通过一般顺序流、条件顺序流、默认顺序流等流对象共同规定。BPMN流程的运行可以视为存在一个表示流程当前运行状态的令牌，从流程中的开始事件出发，沿着顺序流前进。当令牌运行到一个任务时，其会根据任务类型执行相应操作。例如对于样例中的脚本任务，流程引擎将会执行该任务中的脚本；对于用户任务或服务任务，流程引擎将会向链下抛出事件，以调用外部对象或服务完成任务。控制流的分支通过排他网关或并行网关实现，以示例中的流程为例：将排他网关作为起点的顺序流共有两条，分别是一条条件顺序流和一条默认顺序流。在条件顺序流的配置界面，设置了含有判断条件的表达式，即判断表单文件中的简历字段（字符串类型）总长度大于300。当令牌运行到排他网关时，会根据表达式内容在BPMN流程全局区寻找form\_length变量，并对其resume属性进行校验。根据表达式运行结果，令牌会沿着不同路径继续前进，以实现整个BPMN流程的控制流逻辑。



图 18 BPMN控制流表示方式示意图

# 金融领域DSL

## 需求分析

随着科技创新和金融创新的深度融合，金融科技正在成为金融发展的新动力，并为实体经济的发展注入源源不断的新活力，为普惠金融和数字经济的发展提供新动能。然而，金融科技所创造的新的商业模式和金融产品，也为金融市场带来了新的风险。为了防范系统性金融风险，监管部门逐步收紧了金融监管，提出了更多的监管要求，使得金融机构面临的监管压力与合规成本日益增加。如何运用使用新兴信息科技满足金融机构的监管和合规性要求成为重要的领域需求。我国金融监管部门也在积极建立监管科技标准体系，并开展监管科技试点应用，例如中国人民银行印发的《金融科技发展规划（2019-2021年）》中指出，要提升穿透式监管能力，加强监管科技应用，建立健全数字化监管规则库，保证监管信息的真实性和时效性。

监管与合规规则的数字化表达是金融监管与合规领域需要解决的核心技术问题之一。数字化规则是以技术标准和代码协议为基本框架，用以约束和规范对象合规行为的技术规范，在一定程度上具备法律规制的功能[2]。在金融科技的应用中，应当事前建立数字化规则库，使监管与合规规则成为机器可读、可执行、可对接的监管代码；事中采集金融机构监管数据，利用数字化监管规则实现实时穿透式监管；事后利用监管规则分析结果进行风险干预，提升金融监管的有效性。

随着区块链技术和生态的快速发展，监管科技与区块链的关系愈发紧密。利用区块链推动监管科技的创新已经成为了国内外关注的重点之一。国际金融协会（IIF）在报告*RegTech in Financial Services: Technology Solutions for Compliance and Reporting*中指出，区块链可以为金融机构提供透明可信的交易平台与信息共享平台，在数据管理、身份验证等诸多方面助力监管科技。新加坡金融管理局（MAS）在报告*Singapore’s FinTech Journey – Where We Are, What Is Next*中指出，区块链正在被应用于诸多金融场景中，在交易凭证、审计追踪与反洗钱等方面展现出诸多优良特性。欧盟委员会在*FinTech: A More Competitive and Innovative European Financial Sector*中指出，分布式账本（DLT）在资产管理与追踪、实施监管限制、自动化监管与合规等方面具有重要应用。英国金融行为监管局（FCA）开展了面向数字化监管与合规的区块链关键技术研究项目（Blockchain technology for Algorithmic Regulation And Compliance，简称BARAC）和数字监管报告项目（Digital Regulatory Reporting，简称DRR），提出设计领域特定语言实现监管规则的数字化，并在区块链上通过自动执行的智能合约帮助监管部门提高监管效率，提升监管的可靠性和透明性。在我国，国务院出台了《关于加强和规范事中事后监管的指导意见》（国发〔2019〕18号），其中指出要充分发挥现代科技手段在事中事后监管中的作用，依托区块链等新技术推动监管创新，努力做到监管效能最大化、监管成本最优化、对市场主体干扰最小化。此外，中国人民银行、中国证监会等主要金融监管部门也在工作会议上强调利用区块链等技术加强数字监管能力建设，构建新型监管模式。

博雅正链已就金融领规则数字化开展落地实践。博雅正链中国互联网金融协会指导下建设的基于区块链的数字函证平台，成功入选中国人民银行金融科技创新监管沙盒；博雅正链与北京股权交易中心共建的基于区块链的股权登记挂牌及交易系统，应用在股权登记、挂牌、交易，到退出、监管、风控、清结算等多方面，成功入选中国证监会首批资本市场金融科技创新试点。

基于上述落地实践，本课题总结了当前区块链技术在推动金融规则数字化领域面临的技术难点与瓶颈。当前区块链平台在实际应用中主要起到数据存证和交换的作用，监管仍然主要依赖于准入机制和事后的分析、追责；目前的智能合约主要以业务功能为主，没有或只有少量的监管规则嵌入在业务逻辑中，并且与业务强耦合，监管要求更新时难以及时对智能合约中的监管逻辑进行升级。因此，现有的区块链技术难以对数字化监管形成有效支撑。基于区块链构建数字化监管模式，需要满足以下关键技术需求：

（1）降低监管规则数字化的开发难度。现有的智能合约编程语言主要以面向对象语言为主，难以高效地实现相关应用场景中的链上规则数字化表达需求。以Solidity为代表的智能合约编程语言的主要设计目标是实现智能合约中的业务逻辑，由于编程语言的复杂性和程序逻辑验证的困难性，导致了只有专业的编程人员才能够编写智能合约，领域专家难以直接将特定规则转化为智能合约，并且难以验证智能合约的正确性。

（2）业务逻辑与监管规则分离。通过智能合约实现监管规则，能够提高监管的公开性和公平性。但是，在区块链上实现业务逻辑的智能合约和实现监管规则的智能合约应当分别由从业机构和监管部门开发和维护，以保证对不同从业机构监管要求的一致性，并能即时更新监管规则。

（3）保证监管规则的正确性。通过形式化方法对监管规则进行分析和验证能够保证监管规则的正确性，有助于减少监管摩擦，提高监管效率。在监管规则开发过程中和上线部署之前，应当通过形式化工具对监管规则进行检查，避免数字化监管规则中出现编程错误，影响区块链的正常运行。

针对上述领域需求，本课题开展了面向金融领域规则数字化的领域特定语言研究，设计了领域特定语言RegLang，该语言具备可交互性、可演算性和可并发性，便于领域专家编写数字化的监管规则，并在区块链上以智能合约的形式运行，以推动链上规则数字化、自动化、智能化。

## 国内外相关研究工作

为了提高金融领域规则数字化的便利性，国内外相关领域的研究人员已就金融领域的DSL设计开展探索。本节对面向金融领域规则数字化的智能合约编程语言的相关研究工作进行介绍，主要包括面向监管与合规的编程语言和智能合约编程语言两个部分。

### 面向监管与合规的编程语言

国外针对监管规则数字化问题的研究起步较早，研究人员通过设计编程语言描述法律文本，进而应用在信息系统和业务数据的合规检查等方面；我国目前针对这一问题的研究尚处于初步阶段，主要是利用语义挖掘、统计分类等方法分析裁判文书等数据，但对监管规则数字化的研究仍在探索。从研究方法上看，针对规则数字化的研究主要包括：利用建模语言（Modeling Language）[[[2]](#endnote-1)]对法律和监管政策建模，为信息系统需求规范的合规性提供模型化支持；利用受限自然语言（Controlled Natural Language，简称CNL）[[[3]](#endnote-2)]重新编写法律和监管政策，实现法律文本的可读性与可计算性的平衡；利用规则引擎的领域特定语言或通用的逻辑编程语言翻译监管规则，既保证信息系统中业务逻辑与监管规则的分离，又满足业务系统实时检查业务合规性的需求。

Nòmos[[[4]](#endnote-3)]是由Ingolfo等提出的图形化建模语言，通过面向目标的需求工程技术对软件系统需求进行建模，解决软件需求的合规性问题。Nòmos利用分析法学中的霍菲尔德权利理论构建了元模型，并设计了语言的表达和推理方式，能够将软件系统的需求建模推理，判断需求是否合规。通过Nòmos构建的模型具有以下性质：可用性，能够针对用户需求进行合规检查；可审计性，能够审计并监控运行中的系统的需求；可追溯性，能够溯源到最初的合规选择和原始法律。研究人员通过Nòmos构建了美国HIPPA法案等多个法律的模型，并在沙盒场景中对模型的可用性进行了测试。

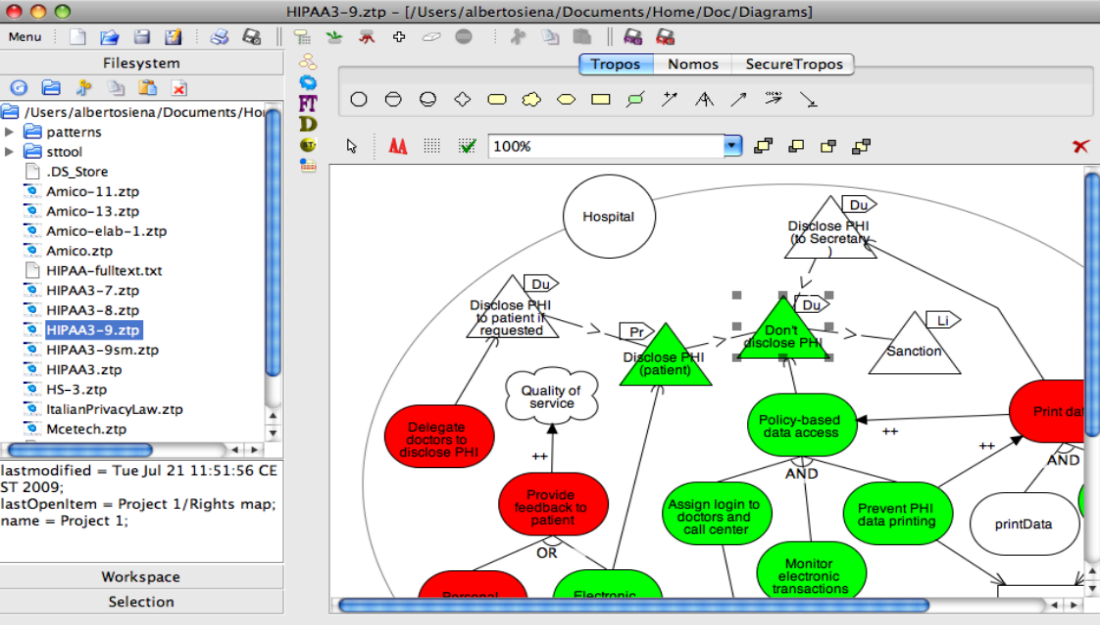


图 19 利用Nòmos对HIPPA法案建模示意图

业务词汇和业务规则语义（Semantics Of Business Vocabulary And Business Rules，简称SBVR）[[[5]](#endnote-4)]是对象管理组织（Object Management Group，简称OMG）定义的业务词汇表，用于保证在组织之间和软件工具之间交换业务词汇和业务规则语义的一致性。SBVR提供了结构化的受限自然语言模型SBVR Structured English（简称SBVR SE）。SBVR SE是自然语言的受限子集，通过预设的关键字连接自然语言短语并定义短语之间的关系。在SBVR SE的基础上，Kholkar等实现了从受限自然语言到DR-Prolog[[[6]](#endnote-5)]的自动转换，实现监管规则的自动推理。由于受限自然语言是自然语言的子集，因此业务专家学习成本较低，并且保证了计算机的可读性，因此被认为是实现法律数字化的可行方案之一。但是由于受限自然语言的工作量十分庞大，并且难以形成统一的标准，因此目前在实际应用中较为少见。

### 智能合约编程语言中的规则数字化

针对区块链上数字化规则的表达问题，国内外已有部分研究工作与产业应用。Findel[[[7]](#endnote-6)]是一种说明式的DSL，便于将金融衍生品的合同翻译为智能合约，并通过以太坊合约对Findel合约进行管理。Seijas等提出了面向金融合约设计的智能合约编程语言Marlowe[[[8]](#endnote-7)]，该语言能够通过Isabelle定理证明器进行形式化验证，确保资金的安全性。Bartoletti等[[[9]](#endnote-8)]提出了BitML，用于在比特币平台上编写智能合约，能够在不依赖于可信第三方的情况下保障两方按照合约要求完成转账。另外，Ciatto等[[[10]](#endnote-9)]也提出了在区块链上利用逻辑编程语言编写智能合约的设想，通过逻辑编程语言编写智能合约，能够提高法律文本与智能合约代码的一致性，并提高智能合约代码的可读性和智能合约行为的可控性。目前已有的金融合约DSL的设计目标主要是便于业务人员在区块链上实现金融业务逻辑，但是较少考虑了金融合约的监管与合规问题，也难以实现监管规则的数字化。

区块链平台EOS也支持了一种受限自然语言作为智能合约，即李嘉图合约（Ricardian Contract）[[[11]](#endnote-10)]，要求开发者同时部署李嘉图合约和代码编写的智能合约，并通过李嘉图合约明确定义使用合约的目的和边界，对供应商和用户的使用进行合法的限制。

TOKENNAME = XYZ  
INFLATION = 5%  
INITIALISSUE = 1,000,000,000  
   
The Community hereby creates a currency known as {{TOKENNAME}}, possession of which is evidence of a contribution to the community. The quantity of {{TOKENNAME}} shall increase no more than {{INFLATION}} per year after the first {{INITIALISSUE}} of {{TOKENNAME}} are distributed.

图 20 李嘉图合约代码示例

Astigarraga[[[12]](#endnote-11)]等提出了面向Hyperledger Fabric设计的规则引擎及规则语言Business Collaboration Rules Language（BCRL），但是BCRL的主要设计目标是根据规则对区块链的账本数据进行更新，无法对区块链上的事务进行实时检查，因此难以应用在监管与合规场景中。FCA开展了面向数字化监管与合规的区块链关键技术研究项目（Blockchain technology for Algorithmic Regulation And Compliance，BARAC）和数字监管报告项目（Digital Regulatory Reporting，DRR），提出设计领域特定语言UmLang实现监管规则的数字化，并在区块链上通过自动执行的智能合约帮助监管部门提高监管效率，但是迄今为止仍未提出完善的设计与实现方案。

## 设计目标

RegLang是一种为在区块链上编写基于规则的监管合约而设计的领域特定语言。为了便于监管专家将监管规则数字化，RegLang在保证对监管规则具有充分的表达能力的基础上，被设计得尽可能简单，并在类型系统、内置函数、逻辑量词的支持方式等方面提高数字化规则编写的便利性。本节从设计目标、语义模型、程序结构、语法规范、类型系统，运算符和内置函数等方面对RegLang语法设计进行详细介绍。

RegLang的设计目标主要是便于领域专家将自然语言编写的领域规则翻译为计算机可执行的数字化规则，并在区块链上以智能合约的形式运行。为了解决现有智能合约编程语言编写难度大、合约耦合强、运行效率低等问题，RegLang在保证语法简洁、便于编写的基础上，应具备可交互性与可演算性，具体定义如下：

（1）可交互性。RegLang的主要功能是对业务合约的事务进行监管，因此在RegLang中应具备与业务合约事务、状态等相关的关键字或变量，使监管合约能够读入业务合约及事务相关的数据并进行计算，但同时不破坏业务合约事务的原子性。

（2）可演算性。RegLang的计算过程本质上是通过逻辑演算验证业务合约的事务是否符合监管与合规规则，因此RegLang应当支持逻辑量词（全称量词和存在量词）和部分逻辑联结词（否定联结词、合取联结词、析取联结词）。

## 语义模型

金融领域的监管与合规规则可分为强制性规定和禁止性规定，强制性规定是指应为某种行为的规定，禁止性规定是指禁止为某种行为的规定。强制性规定是命令从业机构或个人履行特定义务的规范，一般对应法律法规中“应当”、“必须”等条款；禁止性规定是禁止从业机构或个人采用特定模式的规范，一般对应法律法规中“不得”、“禁止”等条款。参考需求工程的定义方式，我们将监管问题进行如下定义。首先，我们将每个监管规则定义为一个应被满足的命题，监管规则分为监管范围部分和监管要求部分，即：

强制性规则集....和禁止性规则集定义如下：

监管规则集为强制性规则集和禁止性规则集的并集：

根据监管规则集，对符合监管规则的状态集的定义如下，其中是能够描述从业机构或个人行为的状态，即对监管规则中命题变元的指派：

例如，考虑《金融机构大额交易和可疑交易报告管理办法》（中国人民银行令〔2016〕第3号），金融机构应当报告人民币5万元以上（含5万元）的现金收支。在仅考虑现金收支的情况下，可以进行如下定义：

在仅考虑这一条监管规则的情况下，。对于一笔现金收支，交易金额可能为5万元以上（含）或5万元以下（不含），金融机构提交大额交易报告的状态可能是“已提交”或“未提交”。因此可能存在四种状态：

其中使的前件和后件都为真，因此使 成立；和使的前件为假，因此和使 成立；而使得的前件为真、后件为假，因此使不成立。综上，、、是满足监管要求的，即；而不满足监管要求，即。

在监管问题的状态集定义中，需要考虑从业机构或个人的行为状态集，以及根据监管规则集定义的状态集，那么监管问题的状态集可定义为：

从软件工程的角度，信息系统的需求问题可定义为对的证明，即在领域知识的条件下，信息系统的规范应当满足需求。类似地，对监管问题也可以进行如下定义，在领域知识的条件下，从业机构或个人的行为应当满足监管规则的要求，即。

当一个状态属于时，则该状态为从业机构或个人的行为。当一个状态属于 时，则该状态在领域知识K的条件下满足监管规则要求。那么可以进行如下定义：当且仅当从业机构或个人的行为状态集中任意一个状态，都满足属于时，从业机构或个人的行为是符合监管规则的，即：

在实际应用中，代表从业机构或个人的行为，监管部门可以根据监管范围和条件，通过行政命令或行业标准对上报数据的格式、内容等进行要求；代表领域知识，例如黑白名单、汇率、额度等；代表数字化的监管规则需要定义的内容。

综上所述，监管问题能够转化为证明是否成立的问题，证明方法为在领域知识的条件下，验证，是否成立，即验证，是否成立。

## 程序结构

RegLang程序结构包含任意数量的知识库代码块和规则代码块。每个代码块以关键字knowledgebase或rule以及代码块的名称开头，以关键字end结尾。解释RegLang文件的顺序为首先解释所有知识库代码块，然后依次解析规则代码块。知识库、规则、注释代码块的详细设计说明如下。

1. **知识库代码块**

知识库维护规则引擎的所有外部知识。每个知识库可以包括多个知识。知识的命名空间为所在的知识库，也就是说不同知识库中相同名称的知识是不同的。

在下面的例子中，knowledgebase(black1).blacklist定义了一个黑名单机制，其中，地址"0xBEeFbeefbEefbeEFbeEfbEEfBEeFbeEfBeEfBeef"被列为黑名单。

knowledgebase black1

knowledge blacklist =["0xBEeFbeefbEefbeEFbeEfbEEfBEeFbeEfBeEfBeef"];

end

图 21 RegLang知识库代码块示例

1. **规则代码块**

规则代码块维护所有的监管规则。一个规则必须包含一个监管范围和至少一条require或prohibit语句。当表达式的值为真时，require语句是被满足的；当表达式的值为假时，prohibit语句是被满足的。一旦有一个语句未被满足，规则引擎就不再计算其他的语句和规则。在下面的例子中，prohibit 1==1;是不被满足的，因此RegLang将不再计算require 2==2;，而立刻返回结果。

rule example  
reg true:  
require 1==1;  
end  
  
rule example  
reg true:  
prohibit 1==1;  
require 2==2;  
end

图 22 RegLang规则代码块示例

1. **注释代码块**

RegLang支持单行注释(//)和多行注释(/\*...\*/)。

// This is a single-line comment.  
  
/\*  
This is a  
multi-line comment.  
\*/

图 23 RegLang注释代码块示例

下图中给出了RegLang的一个简单示例。例子中的前三行代码在black知识库中添加了一条知识blacklist，其他代码创建了一条规则CheckTransfer。在规则部分，第7行是一条强制性规定，表示监管规则要求事务参数\_value的值应不大于 1000000 ；第8行是一条禁止性规定，表示如果事务的发起方或转账的接收方在黑名单中，事务就会被拒绝。

knowledgebase black  
knowledge blacklist = ["0xBEeFbeefbEefbeEFbeEfbEEfBEeFbeEfBeEfBeef"];  
end  
  
rule CheckTransfer  
reg contract(tx.to).name=="ERC20" and tx.function=="transfer":  
require tx.args.\_value <= 1000000;  
prohibit tx.from in knowledgebase(black).blacklist or tx.to in knowledgebase(black).blacklist;  
end

图 24 RegLang程序示例

## 语法设计

RegLang基于Antlr[[[13]](#endnote-12)]实现词法、语法分析、代码翻译与转义。下图展示了RegLang的语法规则定义。

<Reglang> : {<KnowledgebaseBlock> | <RuleBlock>}\*

<KnowledgebaseBlock> : 'knowledgebase' NAME {<KnowledgeDecl>}+ 'end'

<KnowledgeDecl> : 'knowledge' NAME '=' <ArithExpr> ';'

<RuleBlock> : 'rule' NAME <RegScopeStmt> {<RegRuleStmt>}\* 'end'

<RegScopeStmt> : 'reg' <LogicExpr> ':'

<RegRuleStmt> : 'require' <LogicExpr> ';'

| 'prohibit' <LogicExpr> ';'

<LogicExpr> : <ArithExpr> 'in' 'knowledgebase' '(' NAME ')' '.' NAME

| <FuncSet1> '('<ArithExpr> ',' <LogicExpr>')'

| <FuncSet2> '(' <LogicExpr>')'

| <ArithExpr> <ArithBinOp> <ArithExpr>

| <LogicExpr> <LogicBinOp> <LogicExpr>

| '(' <LogicExpr> ')'

| <Boolean>

<ArithExpr> : <FuncSet3> '(' <ArithExpr> ')'

| <FuncSet4> '(' <LogicExpr> {',' <LogicExpr>}\* ')'

| <ArithExpr> <ArithBinOp> <ArithExpr>

| '(' <ArithExpr> ')'

| <Array> | <Var> | NUMBER | STRING

<Var> : 'tx' '.' <TxBasic>

| 'tx' '.' <TxArgs> '.' <Var>

| 'knowledgebase' '(' NAME ')' '.' NAME

| <Var> '[' <ArithExpr> ']'

| '[' <ArithExpr> ']'

| '(' <ArithExpr> ')'

| '(' <Var> ')'

| NAME

<Array> : '[' NUMBER {',' NUMBER}\* ']'

| '[' STRING {',' STRING}\* ']'

<ArithBinOp> : '=='|'!='|'<'|'<='|'>'|'>='|'+'|'-'|'\*'|'/'|'^'|'%'

<LogicBinOp> : 'and' | 'or'

<Boolean> : 'true' | 'false'

<TxBasic> : 'from' | 'to' | 'function'

<TxArgs> : 'args'

<FuncSet1> : 'at\_least'|'at\_most'

<FuncSet2> : 'any\_item'|'all\_items'

<FuncSet3> : 'length'

<FuncSet4> : 'count'

图 25 RegLang中的语法规范定义

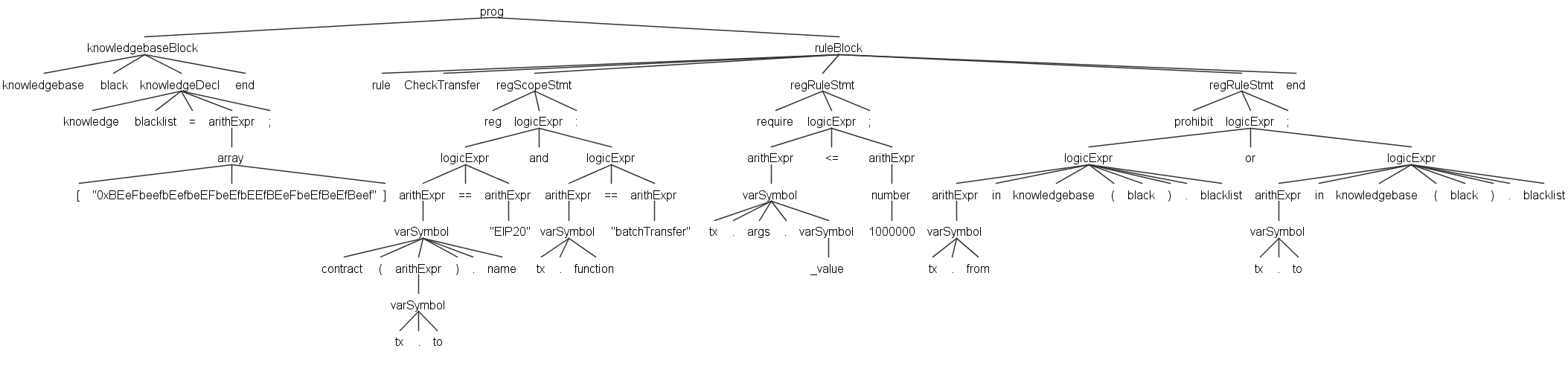
下图展示了“RegLang代码结构”小节中RegLang程序示例所对应的抽象语法树（AST）。

图 26 RegLang程序示例所对应的抽象语法树（AST）

对于RegLang的类型系统，RegLang支持下述基本数据类型。

1. **数值型**

RegLang的数值为无符号整数。数值能够被赋值给一个知识，或者在规则语句中用作常量。

knowledgebase example  
knowledge one = 1;  
end  
  
rule test  
reg true:  
require knowledgebase(example).one == 1;  
end

图 27 RegLang中的数值型变量示例

1. **字符串**

RegLang的字符串由双引号("")包围。字符串能够被赋值给一个知识，或者在规则语句中用作常量。

knowledgebase example  
knowledge str = "This is a string.";  
end  
  
rule test  
reg true:  
require knowledgebase(example).str == "this is a STRING.";  
end

图 28 RegLang中的字符串示例

1. **布尔型**

RegLang的布尔值可以是true和false。布尔值能够被赋值给一个知识，或者在规则语句中用作常量。

knowledgebase example  
knowledge bool = true;  
end  
  
rule test  
reg true:  
require knowledgebase(example).bool;  
end

图 29 RegLang中的bool类型示例

1. **数组**

RegLang只支持数值数组和字符串数组，不支持混合数组。数组只能够被赋值给一个知识。

knowledgebase example  
knowledge numarray = [1, 2, 3];  
knowledge strarray = ["1", "2", "3"];  
end  
  
rule test  
reg true:  
require 1 in knowledgebase(example).numarray and "1" in knowledgebase(example).strarray;  
end

图 30 RegLang中的数组示例

1. **变量**

RegLang中的知识与面向过程的编程语言和Solidity等智能合约编程语言中的变量类似，其与利用BPMN编写的智能合约逻辑共享变量名。知识能够在知识库代码块中进行定义和更新，在规则代码块中使用。示例可参考前面章节。

对于RegLang的运算符和内置函数，RegLang的运算符分为以下四组（优先级从高到低）：

* 算术运算符
* 成员运算符
* 比较运算符
* 逻辑运算符

除了这些运算符以外，为使编写规则更加方便和灵活，RegLang还提供了多个内置函数。

1. **算术运算符**

算术运算符是用来对数值执行常见数学运算的运算符。RegLang中支持下述运算符。

表 2 RegLang中的算术运算符

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 运算符 | 名称 | 优先级 | 示例 | 结果 |
| ^ | 乘方 | 高 | 2 ^ 3 | 8 |
| \* | 乘法 | 中 | 2 \* 3 | 6 |
| / | 除法 | 中 | 5 / 3 | 1 |
| % | 取模 | 中 | 5 % 3 | 2 |
| + | 加法 | 低 | 1 + 2 | 3 |
| - | 减法 | 低 | 2 - 1 | 1 |

1. **成员运算符**

成员运算符in用作检查一个常量或者一个变量的值是否在一个知识中。运算符in的右侧必须是一个数组类型的知识。

knowledgebase black  
knowledge blacklist = ["0xBEeFbeefbEefbeEFbeEfbEEfBEeFbeEfBeEfBeef"];  
end  
  
rule CheckTransfer  
require "0xBEeFbeefbEefbeEFbeEfbEEfBEeFbeEfBeEfBeef" in knowledgebase(black).blacklist;  
prohibit tx.from in knowledgebase(black).blacklist;  
end

图 31 RegLang成员运算符示例

1. **比较运算符**

比较运算符用于比较两个数值或者字符串。RegLang中支持下述比较运算符。

表 3 RegLang中的比较运算符

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 运算符 | 名称 | 支持类型 | 示例 | 结果 |
| == | 等于 | 数值，字符串 | 1 == 1 | true |
| != | 不等于 | 数值，字符串 | 1 != 1 | false |
| < | 小于 | 数值 | 1 < 2 | true |
| <= | 小于等于 | 数值 | 1 <= 2 | true |
| > | 大于 | 数值 | 2 > 1 | true |
| >= | 大于等于 | 数值 | 2 >= 1 | true |

1. **逻辑运算符**

逻辑运算符用于组合逻辑表达式。因为多重否定通常使非开发人员困惑，所以RegLang不支持"not"作为运算符。取而代之的是，RegLang使用require语句和prohibit语句实现逻辑中的“非”，避免了否定词的深层嵌套，并使编写规则更加容易且不易出错。

表 4 RegLang中的逻辑运算符

|  |  |  |  |
| --- | --- | --- | --- |
| 运算符 | 描述 | 示例 | 结果 |
| and | 如果所有语句都是true则返回true | 1<2 and 2<3 | true |
| or | 如果任一语句是true则返回true | 1<2 or 1>2 | true |

1. **内置函数**

RegLang提供了4个内置函数集，分别介绍如下：

函数集1: at\_least() 和 at\_most()。这两个函数都有两个参数：第一个是一个数值，第二个是一个逻辑表达式。表达式的左侧必须是一个数组，通常是tx.args中的数组类型变量。在下面的例子中，at\_least()语句要求tx.args.\_receivers中至少有2项在knowledgebase(white).whitelist中，at\_most()语句要求tx.args.\_receivers中至少有1项在knowledgebase(black).blacklist中。

rule ExampleFunctionSet1  
reg contract(tx.to).name=="EIP20" and tx.function=="batchTransfer":  
require at\_least(2, tx.args.\_receivers in knowledgebase(white).whitelist);  
require at\_most(1, tx.args.\_receivers in knowledgebase(black).blacklist);  
end

图 32 RegLang内置运算示例 1

函数集2: any\_item() 和 all\_items()。这两个函数都有一个参数：参数为一个逻辑表达式。这两个函数与函数集1中的函数类似，对表达式的要求也相同。在下面的例子中，all\_items()语句要求tx.args.\_receivers中的所有项都在knowledgebase(white).whitelist中，any\_item()语句禁止tx.args.\_receivers中的任一项在knowledgebase(black).blacklist中。

rule ExampleFunctionSet2  
reg contract(tx.to).name=="EIP20" and tx.function=="batchTransfer":  
require all\_items(tx.args.\_receivers in knowledgebase(white).whitelist);  
prohibit any\_item(tx.args.\_receivers in knowledgebase(black).blacklist);  
end

图 33 RegLang内置运算示例 2

函数集3: length()。这个函数用于计算数组中有多少项。下面的例子要求tx.args.\_receivers有不超过100项。

rule ExampleFunctionSet3  
reg contract(tx.to).name=="EIP20" and tx.function=="batchTransfer":  
require length(tx.args.\_receivers) <= 100;  
end

图 34 RegLang内置运算示例 3

函数集4: count()。这个函数有任意数量的参数，参数通过逗号(,)分隔。每个参数都是一个逻辑表达式，函数返回表达式为真的个数。在下面的例子中，函数返回2，并且规则是被满足的。

rule ExampleFunctionSet4  
reg contract(tx.to).name=="EIP20" and tx.function=="batchTransfer":  
require count(1==1, 2==2, 1==2) >= 2;  
end

图 35 RegLang内置运算示例 4

## 领域示范应用

### 领域场景

“中国金融科技百强企业”系统是中关村互联网金融研究院推出的金融科技企业评审与管理系统，其通过智能合约评审企业在金融科技方面的业务能力，授予金融科技百强企业。上述智能合约基于博雅领域智能合约编辑软件进行编辑，并使用金融领域智能合约编程语言RegLang以表达金融领域特定业务需求。本系统使用基于可拖拽组件的可视化编程模式，将业务流程的设计过程更贴近业务人员，同时使用金融领域特定语言RegLang对金融领域的监管规则进行数字化，实现金融企业评审与管理的链上运行。

### 示范应用

“中国金融科技百强企业”系统使用金融领域智能合约编程语言RegLang表达金融监管相关的业务需求，对企业的基本情况、运营指标、技术指标等信息使用数字化监管规则进行评审。具体的，对于符合“中国金融科技百强企业”申请条件的企业，系统使用RegLang实现数字化的金融监管规则，将申请过程基于区块链技术来完成，以达到金融企业申请的全流程链上存证。

针对这一场景，本系统需要参选企业提交企业基本信息和技术、运营信息，并将上述信息存储在区块链上。在提交基本信息后，系统使用RegLang模块根据规则预审企业是否属于中关村示范区范围内登记注册的企业、企业持续经营时间是否达到一年、企业发展方向是否为国家以及北京市重点支持的技术领域，是否在经营期间合法合规。若预审不通过，则直接驳回该企业的“金融科技百强企业”申请。若预审通过，则要求企业进一步提交企业的技术和运营信息，包括申请专利数量、产品数量、研究员数量、预估市值、投资规模等若干指标，仍存储在区块链上。这些指标基于数字化规则的链上审核仍通过RegLang实现，判断其是否满足中关村互联网金融研究院的评选规则。若申请企业满足规则，则将该企业的百强申请通过审核并授予证书。若申请企业不满足规则，则将该企业的申请转移到评审专家复审环节，交由链下的评审业务系统并随后发送给评审专家。专家审核完成后，重新将结果提交至区块链上，根据评审结果以决定该企业申请成功或进入候选队列。

金融领域智能合约编程语言RegLang能很好的表达该场景下对金融监管规则进行数字化的业务需求。通过基于可视化智能合约编辑软件和RegLang领域特定语言，能够让业务人员自主实现智能合约主体流程设计并自动生成智能合约代码，降低了业务系统升级过程中的沟通成本和开发成本。该成果可以较好地和现有系统集成，使用期间运行正常，达到了预期效果。

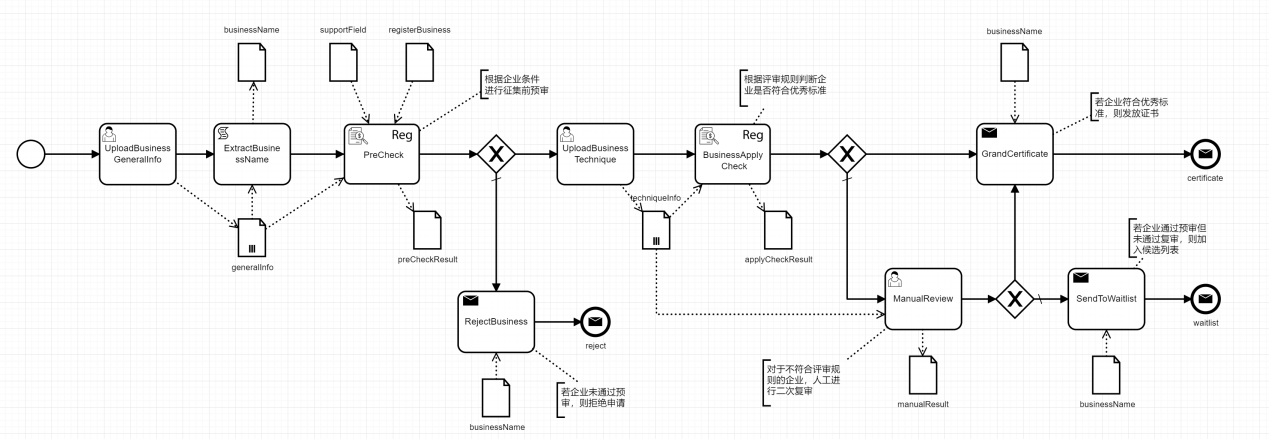


图 36 利用智能合约领域特定语言族编写的金融领域示范应用

# 政务领域DSL

## 需求分析

目前，本课题已针对课题参与各方所积累的政务领域应用场景，开展政务领域需求分析调研。政务智能合约应用以博雅正链数字政府应用和微芯感知的司法存证应用、企业认证应用作为主要分析对象。数字政府应用基于博雅正链的区块链技术打通各个委办局的政务数据，面向医疗机构、卫生健康、感知数据、城市运行等场景提供区块链存证、用证、智能合约、建链、节点对接等服务，实现应用区块链技术提高数字政府的安全性和工作效率、助力可信数字政府建设、推动政务数据共享、协同、高效服务的目标。对于智能合约的需求主要包括政务业务办理的过程记录、流程溯源和记录鉴真，支撑各类链上政务数据应用。

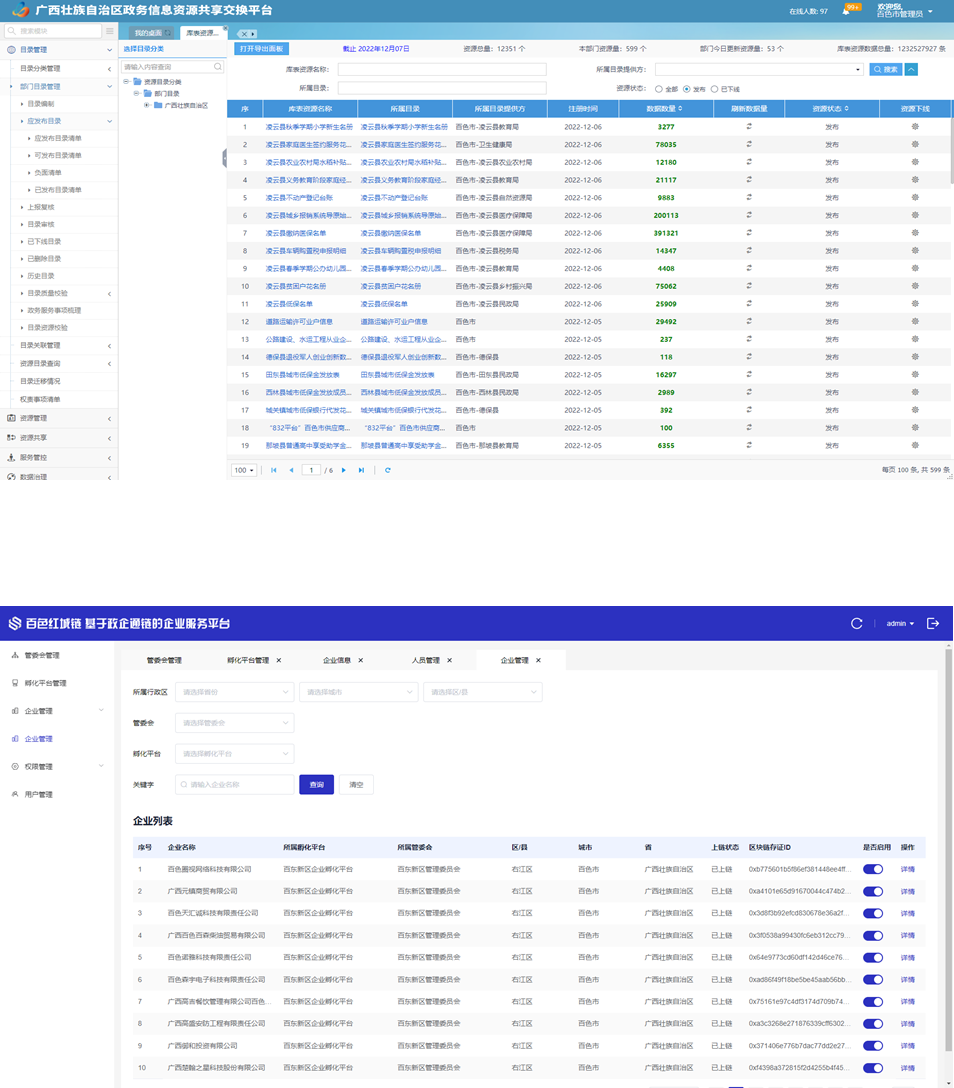
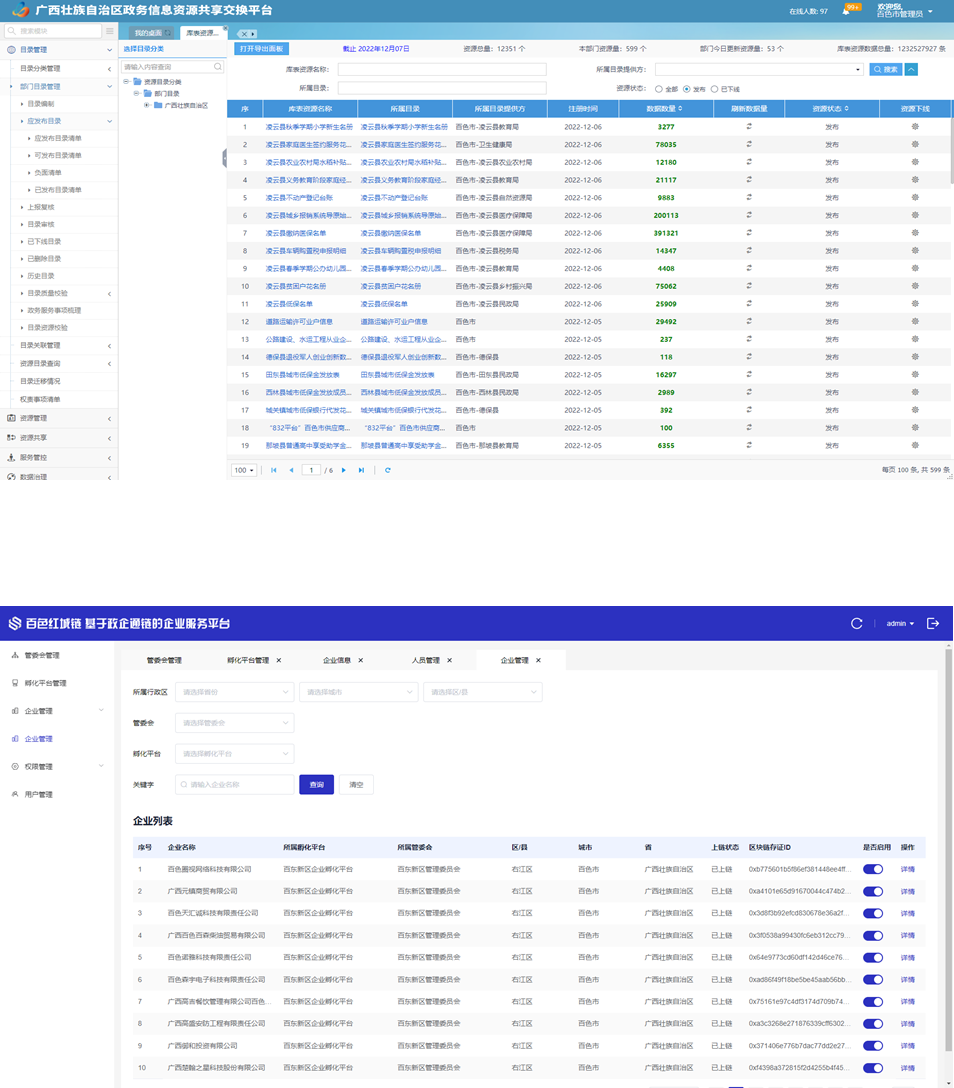


图 37 政务领域应用场景示意图：数字政府应用

司法存证应用为司法业务赋能，能够提高司法效率、降低司法成本，推动完善互联网时代新型证据规则体系。通过长安链多场景及在政务场景的落地，提升运用数据服务经济社会发展的能力，推动政务数据开放利用。长安链已与创业黑马签约，建设朝阳区数字人知识产权可信存证平台，使用GCL和DSL智能合约实现音频、视频、图片、文档等电子数据信息通过智能合约上链、校验、查询等功能，为朝阳区各类互联网3.0和数字人企业在知识产权确权、维权保护、交易护航等过程提供知识产权保护、知识产权取证、知识产权维权等公证法律服务，助力朝阳数字人产业基地的建设。对于智能合约的需求主要包括电子数据信息上链存证合约、电子数据信息校验合约、电子数据信息查询合约等。

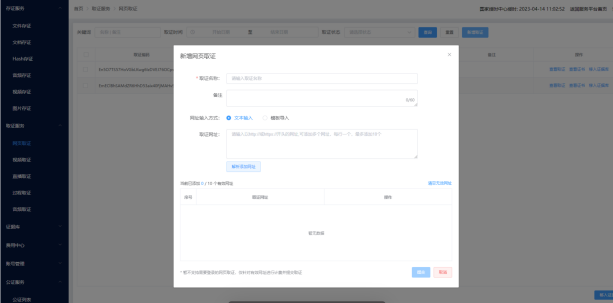


图 38 政务领域应用场景示意图：司法存证应用

企业认证应用依托于区块链和大数据等新技术，为企业提供更好的公共服务，促进政府部门、监管部门、各行企业之间的信息共享和融合，提高企业信息传递效率，降低企业信息造假风险增加企业的可信价值，建立公开透明、公平竞争的市场竞争格局，为企业提供信息安全、可追溯的链式服务，优化市场营商环境、提高经济运行效率、促进资源优化配置。长安链依托在北京市建设的企业认证平台，在企业认证平台中集成智能合约引擎，并使用GCL和DSL智能合约对已有智能合约进行扩充，提供企业信息上链、校验、查询等智能合约维护与管理功能；调整与上层业务的接口，支持上层业务系统通过调用GCL智能合约和区块链进行交互，支持企业认证平台的核心业务功能和系统支撑功能。对于智能合约的需求主要包括信息上链存证合约、信息校验合约、信息查询合约等。

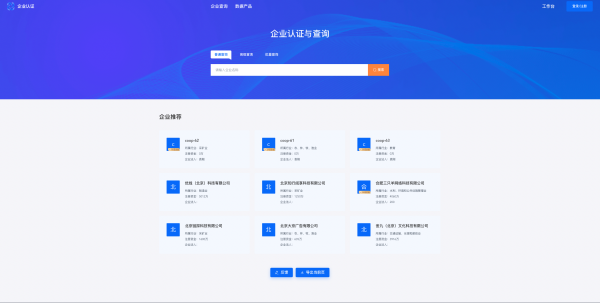


图 39 政务领域应用场景示意图：企业认证应用

在上述应用中，落地实践面临的主要挑战之一是如何在多个参与方中实现有效的访问控制。这些参与方可能包括政府机构、企业、公众以及其他相关组织。每个实体都有其特定的数据访问需求和操作权限。例如，在企业认证等应用场景均设计到多种参与方的协同合作，并可能出现动态的角色变化（如，企业状态从未通过认证变更为通过认证需要经过相关监管部门的授权）。然而，当前智能合约缺乏将访问控制模型转换为链上访问控制规则的有效方式，造成实际应用场景中的开发效率低下，实现流程复杂等问题。而在本报告设计的BPMN中，同样没有内置的访问控制管理模块，使得生成的BPMN依旧依赖于某一中心化第三方进行流程调度与执行，无法发挥区块链所实现的去中心化特性，无法实现多个协同者同时参与的高效流程编排。因此，有必要就实现链上访问控制开展研究，设计用以实现动态角色变换与访问控制的领域特定语言，并以插件形式将其与已有BPMN框架结合，以实现访问控制逻辑的自动代码生成。

## 国内外相关研究工作

### 访问控制模型

自主访问控制模型（DAC，Discretionary Access Control）[[[14]](#endnote-13)]是根据自主访问控制策略建立的一种模型，允许合法用户以用户或用户组的身份访问策略规定的资源，同时阻止非授权用户访问资源。拥有资源权限的用户，可以将该客体的权限分配给其他用户。DAC常见于文件系统，LINUX等操作系统都提供DAC的支持。在实现上，先对用户鉴权，然后根据控制列表决定用户能否访问资源。用户控制权限的修改通常由特权用户或者管理员组实现。DAC最大缺陷就是对权限控制比较分散，比如无法简单地将一组文件设置统一的权限开放给指定的一群用户。在大型组织中，这可能使得DAC难以管理和维护。

强制访问控制模型 (MAC, Mandatory Access Control)[[[15]](#endnote-14)], 是为了弥补 DAC 权限控制过于分散的问题而诞生的。在计算机安全领域指一种由操作系统约束的访问控制，目标是限制主体或发起者访问或对对象或目标执行某种操作的能力。任何主体对任何对象的任何操作都将根据一组授权规则（也称策略）进行测试，决定操作是否允许。在MAC种，每个参与者被赋予一定的安全级别，每个被访问对象也被赋予一定的安全级别，个体能否访问某一对象由双方的关系安全级别决定。

基于角色的访问控制模型（RBAC，Role-Based Access Control）[[[16]](#endnote-15)]是一种在各种应用中广泛使用的访问控制策略。在传统的用户-权限模型中，权限是直接分配给具体的用户的。随着用户数量和权限复杂性的增长，这样的模型将变得难以管理。而在RBAC中，权限是分配给角色的，用户则被分配到这些角色。这样，相同角色的用户可以共享相同的权限，从而简化权限管理流程。下述三元组为RBAC的核心概念：

* 角色（Role）：RBAC模型中的角色可以视为用户的行为和责任的抽象。一个角色代表一组与之相关的权限。
* 用户（User）：系统中的最终实体或操作者，他们将被分配到一个或多个角色。
* 权限（Permission）：代表访问资源的能力。它描述了用户可以执行的操作以及可以操作的资源。

### 面向流程管理的领域特定语言

商业流程管理 (BPM) 包括用于建模、执行、监视、配置和管理处于公司业务核心的流程的方法和概念。这些流程由一组任务组成，这些任务共同实现了一个业务目标。这样一组任务的组成和控制可以以集中的方式编排（choreography）完成（即其中流程所有者在所有时间点都对所有任务具有完全的控制和了解），也可以以分散的方式通过流程编排完成。其中流程控制从流程所有者转移到其他流程参与者，没有一个参与者对流程有完全的控制和了解。

目前，在Web服务领域的国内外相关研究已经针对流程编排与角色管理的领域特定语言开展探索。例如，在业务流程执行语言（BPEL）[[[17]](#endnote-16)]中，通过伙伴链接（Partner Links）支持角色绑定。伙伴链接是一个变量，该变量包含对服务端点的引用。在一个过程的执行过程中，这个变量可以在任何时候被修改。这种方法假设整个过程都是由单一参与者协调的，而这个参与者单方面决定哪个参与者应该绑定到每个角色（即伙伴链接）。然而，这类协议通常难以适用于多参与者协同进行角色绑定的过程。

BPEL4Chor[[[18]](#endnote-17)]设计了一种机制，以扩展BPEL功能。其实现：(1) 参与者行为描述，即每个参与者中的控制流依赖关系，(2) 参与者拓扑，即现有的参与者及其使用消息链接所实现的互相间关系结构，以及(3) 参与者具体配置，即数据格式和端口类型的具体配置。BPEL4Chor允许一个参与者绑定其他参与者到它控制的角色上。但每个角色都是由单一参与者控制的。即：它不支持多协同者的角色绑定，同时， BPEL4Chor不支持角色重新绑定。

Lorikeet[[[19]](#endnote-18)]是一种模型驱动的区块链链上流程管理工具，其扩展了BPMN 2.0，以支持在BPMN过程模型中表示和与注册表交互。该扩展包括两个新元素，RegistryReference和ActionInvocation，它们包括新的图形符号和新的XML属性。RegistryReference表示区块链上的资产数据存储，ActionInvocation表示要调用的资产注册表操作。在注册表方面，Lorikeet的注册表模型提供了一个表单，供用户填写注册表模型输入，具体包括基本信息（注册表名称、描述、用户定义的数据字段及其类型）、注册表类型、基本操作（为每个记录创建/读取/更新/删除和存在检查）和高级操作（记录生命周期管理和外键）。Lorikeet通过注册表的生命周期管理操作上实施访问控制策略，例如创建、更新、删除和转移所有权。其允许单个注册表记录生命周期操作由过程实例（即智能合约）管理。对于从过程实例到注册表的操作调用，只有在确定执行过程实例后，注册表记录的更改才会最终确定：过程实例调用注册表合约函数。然而，Lorikeet是面向资产管理的，而非面向通用的业务合作流程。同时，Lorikeet种所有访问控制流程是静态的，即所有角色分配和权限分配都要在系统初始化时完成，且在系统执行过程中保持不变。

## 设计目标

本部分介绍政务领域领域特定语言AuthLang，拟实现针对政务领域复杂访问控制逻辑的灵活管理与准确表达。AuthLang将基于插件形式继承于BPMN框架中，以将BPMN流程编排为由不同角色参与方所实现的子流程。功能性上，AuthLang将实现多管理者的动态角色绑定与解绑操作，实现复杂流程执行过程中的高效角色管理，提高开发效率。

## 语义模型

1. **静态角色绑定**

AuthLang中的角色模型支持初始化时的静态角色绑定操作，其支持在流程实例初始化时实现对于固定变量的角色分配。例如，下图定义了两个角色，“A”与“B”，其含义为地址在列表[0x1,0x2]中的用户将在函数初始化时获得角色“A”， 地址在列表[0x1]中的用户将在函数初始化时获得角色“B”。在AuthLang中，用户与角色为多对多关系，即一个用户可以由多个角色，一个角色可以由多个用户共同持有。

静态的角色绑定适用于流程中各方权限结构在流程运行中不会发生动态变化的情况。例如，在政务数据共享等应用场景中，各参与方（例如，各政府机构、基层职能部门）之间的角色通常固定，因此可以在流程初始化时对于数据共享的参与各方统一设定角色与权限间的对应关系，并将这类关系硬编码至生成的智能合约中，实现去中心化的访问控制。

Role "A" = [0x1,0x2];

Role "B" = [0x1];

图 40 AuthLang静态角色绑定示例

1. **动态角色管理**

AuthLang中的角色绑定模型允许三种动态角色管理操作，分别为动态权限检查，动态权限分配，动态权限释放。

动态权限检查操作实现的是对于用户所访问资源的权限检查，其以require为关键词，后面连接角色判定语句。示意如下：其含义为只有当msg.sender具有角色B时，才能继续执行，否则，该流程将直接被拒绝。在BPMN中的一个子流程可以拥有多个权限检查语句，以对该流程的多个参与方进行检查。

在政务服务领域，各类政务业务通常具有明确的权限管理要求。例如，在政务流程上链过程中，可能涉及到科层制的审批与授权流程等环节，需要智能合约根据交易的发起方进行权限检查后再进行敏感操作的执行。通过编写AuthLang中的动态权限检查语句，领域专家可以高效地编写这类运行时访问控制逻辑，保证政务服务过程中的权限安全。

require msg.sender is "B";

require x is "A";

图 41 AuthLang动态权限检查示例

事实上，动态权限检查与静态权限初始化操作已实现静态的基于角色访问控制模型。为实现动态的基于角色的访问控制模型，AuthLang额外支持了动态角色提名操作与动态角色释放操作。

动态提名操作允许参与者请求将另一个参与者（或自身）绑定到流程实例中的一个角色。其以“nominates”为关键词，并需要对提名者、被提名者、待提名角色进行规约。动态提名操作通常跟在动态权限检查之后，以实现只有拥有特定权限的人才能提名特定角色(即：角色/权限的多层管理)。例如，下图展示了一个简单的提名流程代码，其首先要求地址x已经具有角色“A”,并允许了地址x对于新地址y的角色提名。在该语句执行结束后，新地址y将同样具有角色“A”，并能通过针对角色“A”的后续的权限检查。

require x is "A";

x nominates y as "A";

图 42 AuthLang动态角色提名示例

相反，动态角色释放操作允许参与者请求将另一个参与者（或自身）从某角色中解绑。其以“nominates”为关键词，并需要对提名者、被提名者、待提名角色进行规约。动态角色释放操作通常跟在动态权限检查之后，以实现只有拥有特定权限的人才能释放特定角色(即：角色/权限的多层管理)。例如，下图展示了一个简单的释放流程代码，其首先要求地址x已经具有角色“A”,并允许了地址x对于新地址y的释放提名。在该语句执行结束后，新地址y将不再具有角色“B”，并不能通过针对角色“B”的后续的权限检查。

require x is "A";

x releases z from "B";

图 43 AuthLang动态角色释放示例

AuthLang的动态角色提名操作与动态角色释放操作可以有效实现运行过程中的权限变换，以作为静态角色绑定的有效补充。例如，在企业招投标的应用中，多个投标企业中将有一个企业作为中标企业，进入后续的项目管理流程。而从投标参与方到项目中标方之间的动态角色变化需要由评审委员会等角色进行授权。

在AuthLang与BPMN的集成中，静态角色绑定、动态角色检查、以及动态角色释放与分配将作为三个单独模块进行。具体地，静态角色绑定将在整个实例初始化过程中完成，用于实现流程开始时地角色分配。动态角色检查将在各个子流程执行开始前完成，以实现各个子流程内部地访问控制逻辑。各个BPMN中的子流程可以通过引入动态角色检查语句来转换为实现了内部访问控制逻辑的子流程。动态角色释放与分配则将作为单独的子流程，以显式地表示流程执行过程中所有角色的动态管理规则，他们将作为整个流程执行中的必要步骤来进行实现。

## 语法设计

AuthLang基于Antlr实现词法、语法分析、代码翻译与转义。下图展示了AuthLang的语法规则定义。

grammar role;

prog: roleDefinition EOF| roleOperationList EOF | roleDefinition roleOperationList EOF;

roleDefinition : roleInitialization | roleDefinition roleInitialization;

roleInitialization : 'Role' role '=' '[' accountList ']' ';';

accountList : account | accountList ',' account ;

roleOperationList: roleOperation ';' | roleOperation ';' roleOperationList;

roleOperation: releaseStatement | requireStatement | nominateStatement;

requireStatement : 'require' someone 'is' role;

nominateStatement : someone 'nominates' someone 'as' role ;

releaseStatement : someone 'releases' someone 'from' role ;

someone: var|account;

var: VARIABLE;

account : ADDRESS;

role : STRING;

ADDRESS:'0''x'[0-9abcdefABCDEF]+;

INT : [0-9]+ ;

STRING : '"' (~'"')\* '"';

WS : [ \t\u000C\r\n]+ -> skip;

NEWLINE : [\r\n]+ -> skip;

VARIABLE : [A-Za-z]+[A-Za-z0-9.]\*;

图 44 AuthLang语法规范

每条语法规则的具体含义如下：

prog: 程序的入口点。它定义了整个程序的结构，可以由角色定义（roleDefinition)、角色操作列表（roleOperationList），或者两者的组合构成。程序必须以文件结束符(EOF)结尾。

roleDefinition: 角色定义部分用于声明系统中的角色及其对应的账户列表。它可以是单个角色初始化（roleInitialization），也可以是多个角色初始化的序列。

roleInitialization: 角色初始化规则用于定义一个特定的角色以及与该角色关联的账户列表。语法形式为 Role 角色名 = [账户列表];。例如，Role admin = [Alice, Bob]; 定义了一个名为 admin 的角色，关联了两个账户 Alice 和 Bob。

accountList: 账户列表定义了与特定角色关联的一组账户。它可以是单个账户，也可以是用逗号分隔的多个账户。例如，Alice, Bob, Charlie 是一个包含三个账户的账户列表。

roleOperationList: 角色操作列表定义了一系列对角色进行操作的语句。这些操作包括释放声明（releaseStatement）、要求声明（requireStatement）和提名声明（nominateStatement）。角色操作列表可以是单个角色操作，也可以是多个角色操作的序列，每个操作以分号(;)结尾。

roleOperation: 角色操作是对角色进行操作的具体语句，包括释放声明、要求声明和提名声明。

requireStatement: 要求声明用于指定某个账户必须具有特定的角色。语法形式为 require 账户 is 角色;。例如，require Alice is admin; 表示要求 Alice 必须是 admin 角色。

nominateStatement: 提名声明用于将某个账户提名为特定角色。语法形式为 账户1 nominates 账户2 as 角色;。例如，Bob nominates Charlie as editor; 表示 Bob 将 Charlie 提名为 editor 角色。

releaseStatement: 释放声明用于将某个账户从特定角色中释放。语法形式为 账户1 releases 账户2 from 角色;。例如，Alice releases Bob from admin; 表示 Alice 将 Bob 从 admin 角色中释放。

每条词法规则具体含义如下：

ADDRESS: 该规则定义了地址词法单元。在词法分析过程中，当遇到以 0x 开头，后跟16进制数字（由0-9或a-f或A-F组成）的字符序列时，该序列被识别为一个地址词法单元。例如，字符序列 0x123abc 将被识别为一个地址词法单元。

INT: 该规则定义了整数词法单元。在词法分析过程中，连续的数字字符（0-9）序列被识别为一个整数词法单元。例如，字符序列 123 将被识别为一个整数词法单元。

STRING: 该规则定义了字符串词法单元。在词法分析过程中，以双引号 (") 开头和结尾，且中间不包含双引号的字符序列被识别为一个字符串词法单元。例如，字符序列 "hello world" 将被识别为一个字符串词法单元。

WS (Whitespace): 该规则用于识别空白字符词法单元，包括空格 ( )，制表符 (\t)，换页符 (\u000C)，回车符 (\r) 和换行符 (\n)。在词法分析过程中，这类词法单元被忽略，不会作为 token 传递给解析器。

NEWLINE: 该规则用于识别换行符词法单元，包括回车符 (\r) 和换行符 (\n)。同样，在词法分析过程中，这类词法单元被忽略。

VARIABLE: 该规则定义了变量名词法单元。在词法分析过程中，以字母（A-Z 或 a-z）开头，后跟字母、数字（0-9）或点号（.）的字符序列被识别为一个变量名词法单元。例如，字符序列 varName 和 user.name 将被识别为变量名词法单元。

### 静态角色绑定语法设计

静态角色绑定在AuthLang中通过角色初始化语句实现。这种绑定在流程初始化时确定，之后在整个流程执行期间保持不变。例如，在图40示例中，角色 "A" 被静态绑定到地址 0x1 和 0x2。这意味着在整个流程执行期间，这两个地址都将具有角色 "A" 的权限。角色 "B" 被静态绑定到地址 0x1。这意味着在整个流程执行期间，地址 0x1 将同时具有角色 "A" 和角色 "B" 的权限。

这种静态角色绑定适用于那些权限结构在流程运行中不会发生变化的场景，如政务数据共享等应用，可以在流程初始化时统一设定参与方的角色和权限。

### 动态角色管理语法设计

动态角色管理提供了在流程执行过程中动态调整权限的能力，包括动态权限检查、动态角色提名和动态角色释放。下面将分别介绍三个功能的使用实例：

* 动态权限检查。例如，在图41中，require 关键字用于检查 msg.sender（即当前交易的发起方）是否具有角色 "B"。如果不具有，则流程将被终止。这种动态权限检查适用于需要根据交易发起方的角色动态控制访问权限的场景。
* 动态角色提名。在图42中，首先使用 require 关键字检查地址 x 是否具有角色 "A"。如果满足条件，那么地址 x 可以使用 nominates 关键字将新地址 y 提名为角色 "A"。这种动态角色提名机制允许在流程执行过程中动态地为参与者分配角色。
* 动态角色释放。在图43中，首先使用 require 关键字检查地址 x 是否具有角色 "A"。如果满足条件，那么地址 x 可以使用 releases 关键字将地址 z 从角色 "B" 中释放。这种动态角色释放机制允许在流程执行过程中动态地撤销参与者的角色。

上述实例AuthLang如何在区块链智能合约中实现静态角色绑定和动态角色管理，以支持复杂的访问控制逻辑。静态角色绑定提供了稳定的权限结构，而动态角色管理则提供了灵活的权限调整机制，以适应流程执行过程中的权限变化需求。

### 语法/词法解析实例

下面展示一个AuthLang编写的简单样例，及其对应的抽象语法树。

1. Role "A" = [0x1,0x2];
2. require x is "A";

图 45 AuthLang代码样例

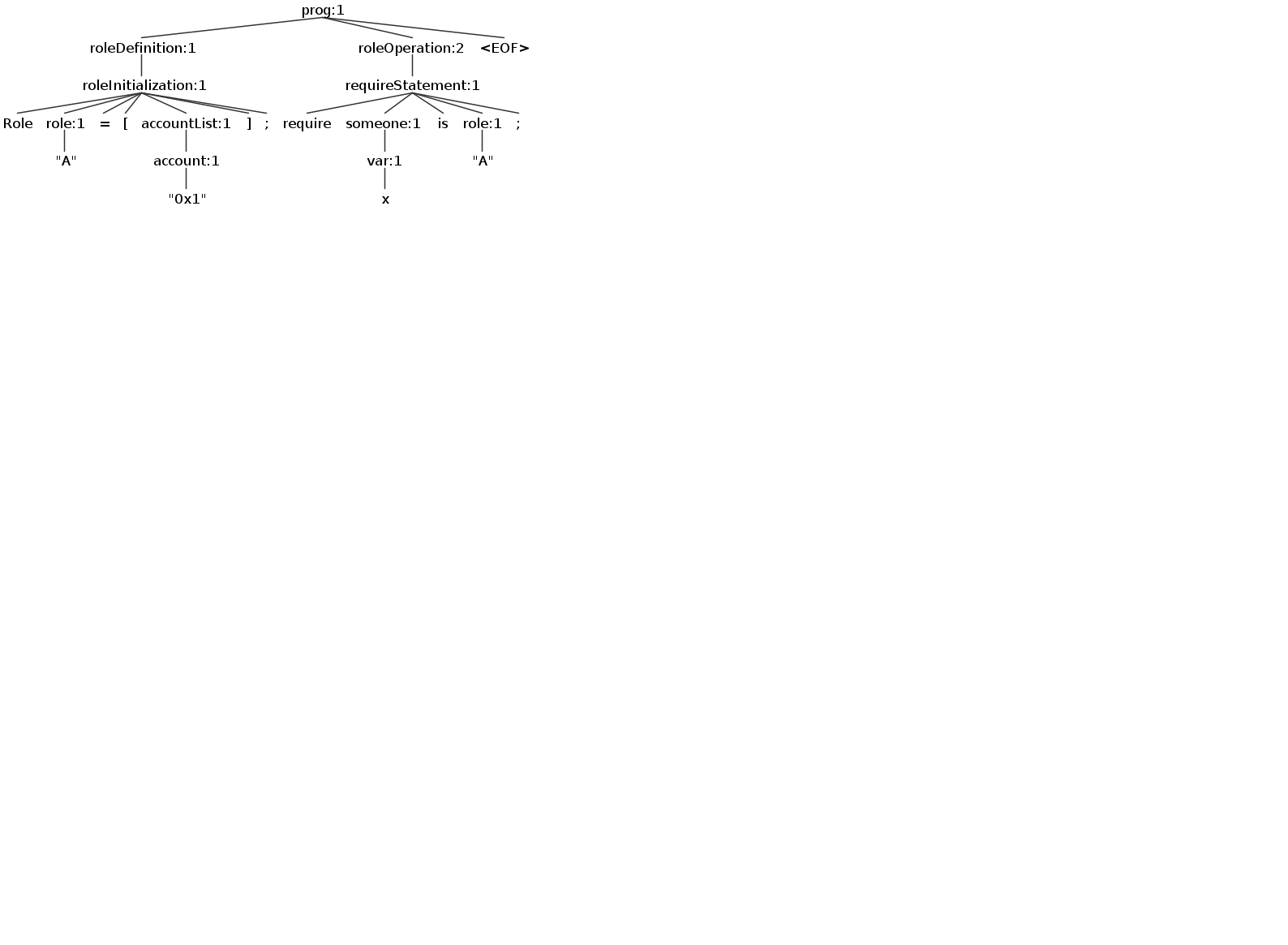


图 46 AuthLang代码样例对应的抽象语法树

## 领域示范应用

### 领域场景

“百色通”是广西省百色市建设的基于区块链的政务服务平台，通过智能合约实现公民电子证照操作记录的安全存储、可信共享、存证追溯。通过在项目中应用博雅正链智能合约领域特定语言族和AuthLang对现有智能合约开发流程进行改造，提高了业务人员和开发人员之间的需求沟通效率，并且能够直接生成智能合约代码，降低了业务系统研发周期和成本，解决了政务服务中申请证照-发放证照-展示证照-验证证照中涉及到的政府部门多、权限复杂、操作敏感等问题，实现政务业务办理的过程记录、流程溯源和记录鉴真。

### 示范应用

百色市大数据发展局是百色市负责统筹推进全市信息化、数字化发展工作的政府单位，其通过建设与牵头运营“百色通”，顺利完成政务服务、生活缴费、交通出行、医疗保障、人才服务、教育服务、旅游购物等领域的数十项便民服务功能。其中，其允许市民用户在“百色通”上传身份信息，并生成身份证件等证照信息，用于政务服务过程中有关部门对市民证照进行快速与便捷核验。具体地，百色市大数据发展局接入区块链系统，以实现电子证照的申请-发放过程的全流程溯源与追踪。

百色市政务服务局是为百色市民提供政务服务的主要单位，在服务过程中，百色政务服务局需要对市民证照进行核验，以进行相关政务业务办理。具体地，百色市政务服务局接受百色市大数据发展局所签发的电子证照，并接入区块链系统，以实现电子证照展示-验证等流程的可信共享与存证追溯。

针对这一场景，本项目基于智能合约领域特定语言族，设计并实现下述智能合约，利用智能合约领域特定语言族中的AuthLang模块管理市民（证照使用方），百色市政务服务局（证照发放方），百色市政务服务局（证照验证方）的角色与流程权限，并通过自动触发链上事件等方式，实现政务业务办理的证照使用/验证过程记录、流程溯源和记录鉴真。该成果可以较好地和现有系统集成，使用期间运行正常，达到了预期效果。

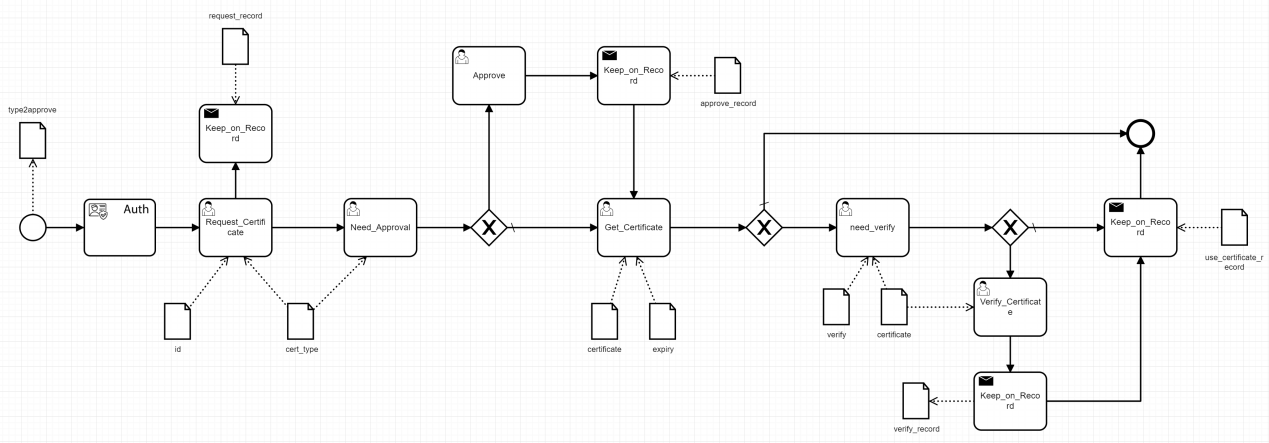


图 47 利用智能合约领域特定语言族编写的政务领域示范应用

# 民生领域DSL

## 需求分析

民生智能合约应用以博雅正链的乡村振兴、医疗数据共享作为主要分析对象。

乡村振兴应用通过整合多部门数据，基于博雅正链的区块链进行部门数据可信存证和共享，按照“标准统一、数据安全、全过程留痕”的原则，构建统一、健全的人口、经济、产业、资源等链上可信基础数据仓库，并进一步以数据比对、数据共享、数据融合等方式，提供数据治理服务。其中，产业兴旺模块面向涉农产业提供产业数据采集和管理功能，生态宜居模块面向多角色提供环境问题发布和推送功能；乡风文明模块提供旅游资源信息综合管理功能；治理有效通过所有关键数据真实上链监督乡村数字化治理的良性发展。博雅正链将依托平果市数字乡村大数据综合服务平台等乡村振兴平台开展GCL和DSL智能合约应用，使用智能合约实现各模块功能，进一步激活三农数据要素活力和价值。对于智能合约的需求主要包括涉农产业数据管理、生态环境问题发布、旅游资源信息管理、乡村数字化治理数据存证。



图 48 民生领域应用场景示意图：乡村振兴应用

医疗数据共享应用针对当前医疗临床科研数据共享存在的“不愿、不敢、不能”问题，采用区块链+分布式统计、联邦学习等技术，将本地、中心式处理转为联邦学习、群体协作模式，达到在数据不出医院的前提下实现跨机构间数据安全共享协同分析，在保证数据隐私的前提下提高医院间的医疗数据安全共享能力，充分发挥医疗数据要素价值。博雅正链目前已联合中国中医科学院中医药数据中心，搭建了基于区块链的中医数据联合循证分析可信联盟链，首期接入中国中医药循证医学中心消化疾病协同创新联盟的14家医院作为第一批优势病种承担单位，对于智能合约的需求主要包括链上任务发布、数据资源管理、数据权限管理、分析结果汇总、分析结果查询、分析过程追溯验证等功能。

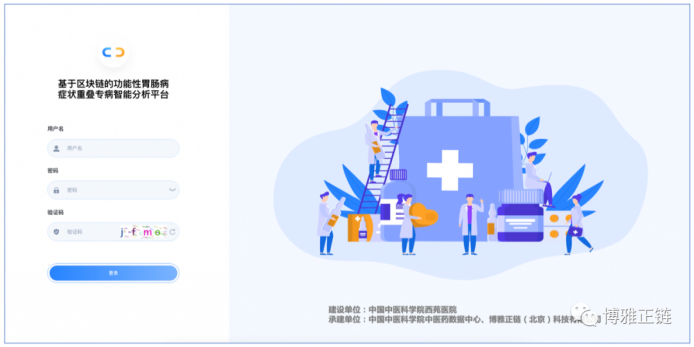
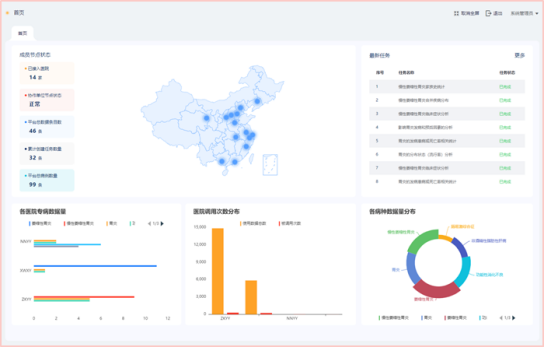


图 49 民生领域应用场景示意图：医疗数据共享应用

## 国内外相关研究工作

为了达到在数据不出医院的前提下实现跨机构间数据安全共享协同分析，在保证数据隐私的前提下提高医院间的医疗数据安全共享能力，充分发挥医疗数据要素价值，国内外相关领域的研究人员已就民生领域的DSL设计开展探索。本节对面向民政领域隐私数据共享的智能合约编程语言的相关研究工作进行介绍。

文章[[[20]](#endnote-19)]引入了名为zkay [zi:keI]的语言。该语言有能力清晰地将私有数据的逻辑和所有权规定与使用NIZK证明来实现此规范的任务相区分。在图1a中，文章展示了一个zkay合约的例子。为了满足第一个任务的需求，zkay经过深思熟虑的设计，支持细粒度、富于表达力且直观的隐私规范。这允许开发者通过为特定账户标注私有变量来明确数据所有权。此外，它设有解密声明，要求开发者明确指出智能合约所揭示的信息内容。文章为zkay合约的数据隐私语义提供了正式定义，以清楚地理解开发者所规定的隐私概念。为了应对前述的挑战，zkay的类型系统整合了隐私类型，从而在静态层面上执行关键属性。首先，此类型系统排除了不可能实现的程序（C1），确保了类型合适的zkay合约能够通过NIZK证明得以实现。例如，它保证函数的私有操作仅基于固定量的私有数据。其次，它限制了操作仅基于调用者可获得的数据（C2），以确保合约功能始终可以实施。再者，通过简单地忽略隐私类型，可以轻松地遵循zkay程序的逻辑（C3）。最后，zkay避免了隐式的信息泄漏，例如，不允许在没有明确解密的情况下，将私人数据写入公共存储（C4）。

文章使用一个示例来说明如何利用zkay来指定隐私约束以及这些规范如何通过NIZK证明转化为合约以进行执行。

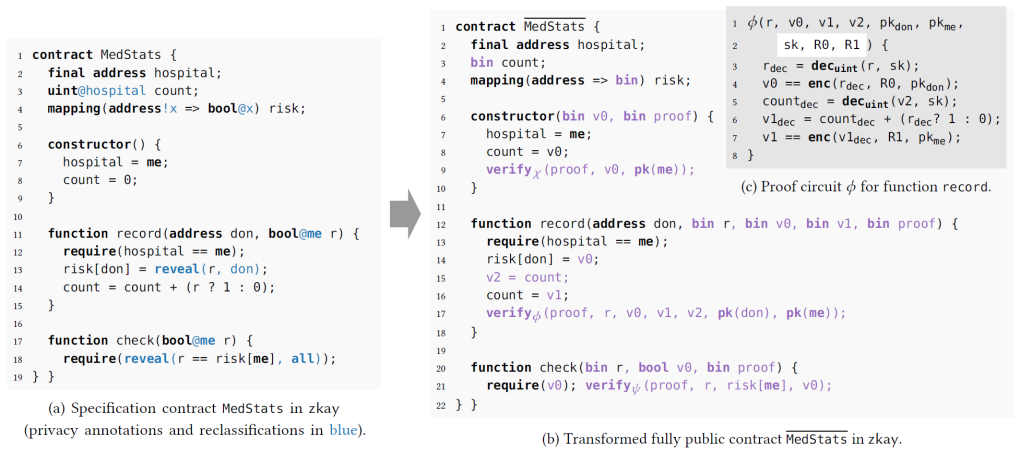


图 50 zkay合约转换流程图

在本部分，我们引用一个示范性的例子，明确阐述如何应用zkay以明确隐私约束，以及如何将这些规范转化为利用NIZK证明进行执行的合约。为了便于那些对区块链或NIZK证明不甚了解的读者，我们在附录A中详细总结了本文所需的关键术语。例证：医疗数据统计。图1a中的MedStats合约使医院能够收集献血者的统计资料。为此目的，医院可以利用record函数记录每位献血者的详细信息。医院作为参数提交献血者的地址及其是否属于风险群体的信息，例如由于旅行史或近期疾病。暂不考虑蓝色隐私标注，record函数（i）在献血者的密钥下的风险映射中记录所提供的数据，并且（ii）仅当献血者属于风险群体时将计数增加一。为核实所收集统计数据的真实性，献血者可以调用check函数，该函数要求risk[me]存储正确值r。在此，“me”代指调用方（与Solidity中的msg.sender相似）。通过观察其他献血者对check的多次成功调用，献血者可以确信统计是准确的。在记录众多献血者的数据后，医院可以公开计数，可能还需采用差分隐私机制进行保护（未在文中展示）。

捐赠者是否属于风险群体是敏感信息，通过zkay的类型系统，我们可以实现对这一信息的保护。

隐私规范。为了保护关于风险群体成员资格的信息，MedStats通过隐私注释来指定隐私限制，确保类型为τ @α（由数据类型τ和隐私类型α组成）的值只能由其所有者α读取。例如，第3行指定count是其所有者医院的私有数据，这意味着只有医院可以读取count。我们注意到，与读操作不同，写操作没有受到限制，因此任何人都可以写入count。与count不同，第2行的hospital没有隐私注释，这意味着它的值是公开的（即，任何账户都可以读取它）。为了强调某个类型是公开的，我们可以明确地标注它为@all。

对于映射，zkay支持细粒度的隐私规范，其中映射条目的所有者可以取决于映射键。例如，第4行将映射risk的键标记为名为x的名称，并在其条目类型bool@x中引用此名称。因此，第13行的risk[don]是don的私有数据。对于嵌套映射，显式地标记映射键尤其有用。也就是说，对于类型为mapping(address!x => mapping(address => uint@x))的m，我们有m[α][β]是α的私有数据。一般来说，隐私注释α可以是(i) me, (ii) all, (iii) 状态变量（即，合约字段）或(iv) 映射键标签。对于情况(iii)，zkay的类型系统确保α的类型是address@all，这意味着所有者是（公开知道的）地址，并且α被声明为final（例如，第2行中的hospital）。zkay的类型系统确保final变量保持不变。这防止了所有权转移，出于简单起见，我们在zkay中不允许这样做。在§9中，我们讨论了如何扩展zkay以支持字段的所有权转移。例如，修改hospital将隐式地导致count获得新的所有者。

ZeeStar[[[21]](#endnote-20)]通过整合zkay与同态加密技术，成功克服了zkay在表达能力上的局限性。同态加密技术使得能够在外部数据上执行特定的运算，其中最为核心的是加法操作。在医疗隐私数据共享领域，这种结合具有巨大的应用潜力。

该整合后的系统，名为ZeeStar，包括一种用于详细描述的高度表现力语言以及一个能够在智能合约中自动执行数据隐私规范的编译器。ZeeStar语言在zkay的隐私注释基础上发展，且进一步允许程序操作外部数据。ZeeStar编译器将NIZK证明与加法同态加密相结合，使得相关程序能够在以太坊平台上运行。通过精妙地结合这两种技术，ZeeStar不仅实现了同态加法，还能够处理涉及多数所有者的乘法运算。这进一步扩展了其在诸如零知识传输等复杂应用中的应用范围。此外，ZeeStar具备将同态和非同态加密方案结合的能力，并在zkay的隐私定义框架下，为其隐私性提供了可证明的保障。

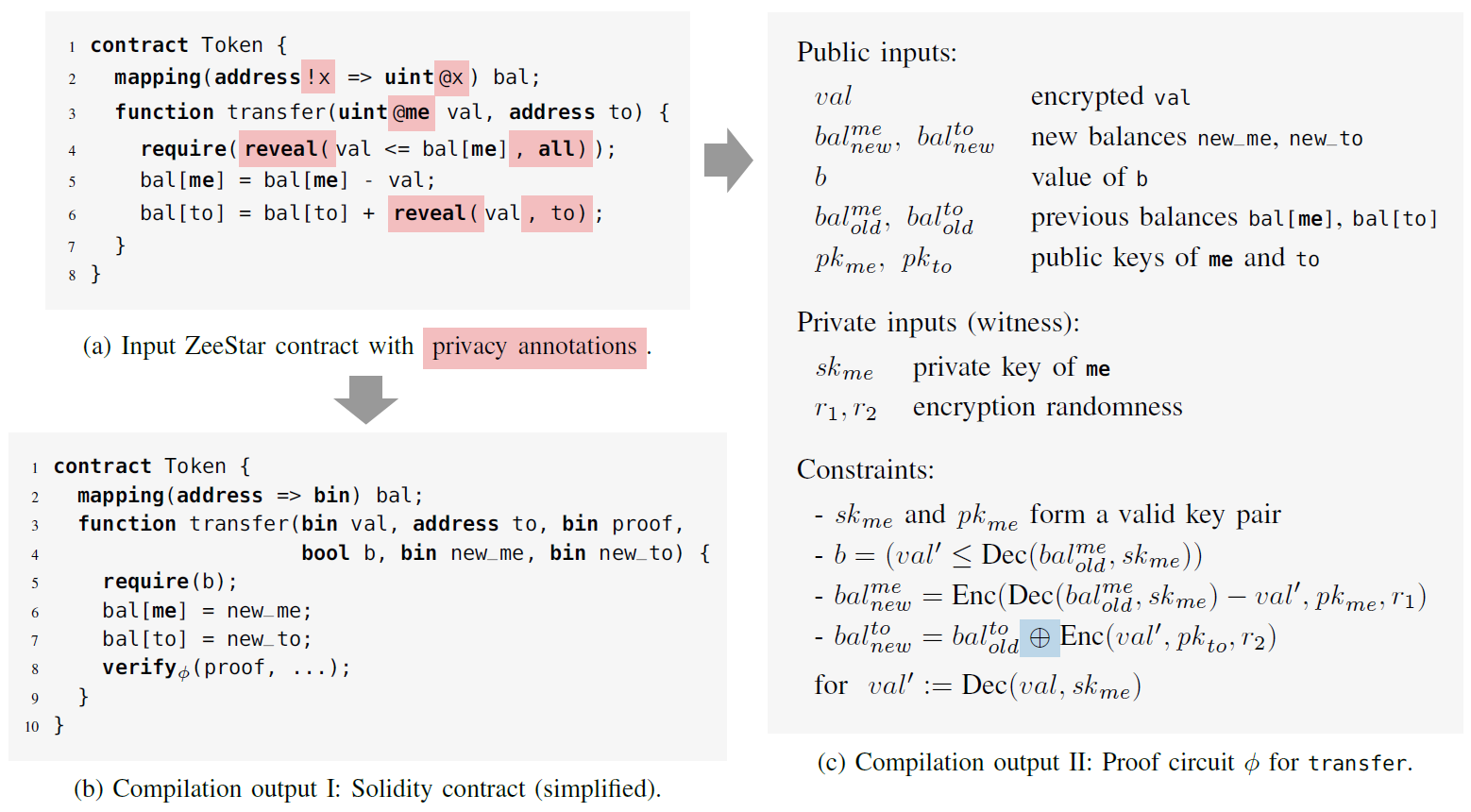


图 51 ZeeStar合约编译流程示意图

## 语义模型

本部分介绍民生领域领域特定语言CryptLang，拟实现针对民生领域复杂密码学逻辑的灵活管理与准确表达。CryptLang将基于插件形式继承于BPMN框架中，以将BPMN流程编排为由不同角色参与方所实现的子流程。功能性上，CryptLang将实现隐私数据的的加密与验证工作，实现复杂流程执行过程中的高效密码学API调用，提高开发效率。

本研究项目以BPMN为基础框架，旨在展现其在特定应用领域的实际运用效果。项目的主要目标是实现BPMN到Solidity的自动化转换过程。此研究所涉及的民生应用案例主要围绕乡村振兴和医疗数据共享两大领域。

虽然BPMN的图形界面能较为有效地满足上述应用的基本语义需求，但在涉及隐私数据共享及统计等核心问题时，如医疗数据共享和旅游信息管理，便需要依托密码学的相关技术，包括数字签名、承诺机制和零知识证明等，以确保其实现。在医疗领域，患者的医疗记录和个人健康信息需维护高度保密性与安全性，而密码学的加密技术则有效保障了这些敏感信息不被未授权访问或泄漏。在乡村振兴方面，涉及的农业数据，如作物产量、气候变化、土壤状况等，同样需借助密码学技术以防止非法访问和篡改，确保信息安全。

鉴于开发人员可能缺乏深入的密码学知识，并未必熟悉区块链提供的底层密码学API，涉及复杂密码学任务的构建过程可能面临潜在的安全风险。

因此，本研究提出了一种专门针对密码学应用而设计的领域特定语言——CryptLang，旨在解决民生领域的隐私保护问题。该语言的设计初衷是为不太熟悉密码学的开发者提供一个简易的操作平台，帮助他们在民生领域轻松完成复杂的密码学任务。CryptLang可以被集成到BPMN的整体架构中，使用户能够在BPMN网页界面的侧边栏直接选择CryptLang任务框，并在其中进行编码以实现相应的密码学任务。值得一提的是，CryptLang不仅适用于民生领域，同样也能在政务或金融领域作为核心的密码学组件发挥作用。

接下来，本文将详细介绍所提出的领域特定语言（DSL）的操作语义。在本节中，我们将专注于描述该语言核心的密码学语义，尤其是针对民生领域中的隐私保护功能，而其他功能则由外部的BPMN框架进行描述和管理。正式定义中，语义配置被明确定义为 𝜇 := (𝑣𝑎𝑟𝑠, 𝑅, 𝑃𝑜𝑜𝑙𝑀, 𝑁𝑐)，其中 𝑣𝑎𝑟𝑠 代表程序变量，𝑅 是验证结果，𝑃𝑜𝑜𝑙𝑀 包含了先前提交的链下证明集合（如签名、哈希前像），而 𝑁𝑐 是签名者的nonce。

程序变量以元组 𝑣𝑎𝑟𝑠 = (𝑀, 𝐻, 𝐼, 𝑃𝑟, 𝐶𝑖𝑑, 𝐴𝑑) 表示，其中 𝑀 ∈ M 为密码学方法，𝐻 ∈ H 为哈希方法，𝐼 ∈ I 包含链上数据，𝑃𝑟 ∈ P 是链下证明，𝐶𝑖𝑑 ∈ C 代表验证合同的区块链标识符，而 𝐴𝑑 ∈ A 是验证合同的地址。

符号 𝜇 = [[𝑎]] ⇒ 𝜇′ 表示在执行操作 𝑎 后，系统状态从 𝜇 转变为 𝜇′。CryptLang的语法与其语义均已整合密码学类别。其中，验证结果 𝑅 是决定后续操作的关键变量。同时，转变后的结果 𝑅′ 与众多变量有关，这包括密码学算法 𝑀，哈希方法 𝐻，链上输入 𝐼，链下证明 𝑃𝑟，池 𝑃𝑜𝑜𝑙𝑀，以及区块链标识符 𝐶𝑖𝑑，地址 𝐴𝑑 和 nonce 𝑁𝑐。

为了直观解释，验证函数 𝑉 𝑅𝐹𝑀 可被划分为两部分：首先验证链上输入 𝐼 与链下证明 𝑃𝑟 的一致性；其次，检查已提交的链下证明 𝑃𝑟 是否已被使用。在签名过程中，nonce 𝑁𝑐 被嵌入到已签名的消息内容中，因此每次成功验证后需要递增相关签名者的地址nonce。在承诺处理中，更为高效的方法是利用𝑃𝑜𝑜𝑙𝑀 作为存储先前提交的 𝑃𝑟 的仓库。随后，验证过程会检查新的 𝑃𝑟 是否已存在于 𝑃𝑜𝑜𝑙𝑀 中。CryptLang编译器将自动执行上述流程。

## 程序结构

CryptLang作为一种领域特定语言（DSL），专门为智能合约中的密码学任务而设计，旨在使开发人员能够轻松掌握并应用其内置的加密特性。当CryptLang集成到BPMN框架中后，它不仅拓宽了密码学操作的范畴，还能在民生领域中高效地处理隐私保护问题。下文将对CryptLang这一专为处理高级密码学任务而定制的工具进行深入探讨，并详尽解释其工作流程以及核心语法和语义。

在民生领域的隐私保护中，链下签名及其在链上的验证是一项关键技术。这种签名验证机制能够实现对隐私数据的有效访问控制。举例来说，以ECDSA在链上的签名验证为例，CryptLang的编译流程可分为几个关键步骤：首先，开发人员使用CryptLang描述加密算法和链上输入数据。随后，编译器通过解析器对CryptLang代码进行分析，生成相应的抽象语法树（AST）。为确保生成的代码安全可靠，系统将参照权威公共存储库（如GitHub上的OpenZeppelin、RSA签名库和验证Merkle证明的库）中的标准密码模板和AST，生成对应的Solidity代码。最终，这些Solidity代码会部署到区块链上，供用户提交并验证其签名。

为更直观展示其应用，以下是一个具体示例：CryptLang帮助医院等医疗记录存储机构快速生成验证签名并提供医疗记录的合约，具体代码如下图所示：

// CryptLang

contract MedicalRecord {

function permit(address \_owner, address \_hospital, uint256 \_year) public {

@ECDSA with SHA3-256 (\_owner, \_hospital, \_year);

\_search(\_owner, \_hospital, \_year);

}

}

图 52 CryptLang程序示例

CryptLang的大部分用法和功能都与Solidity类似，只有用到“@”操作符时才真正使用到CryptLang的功能。建议在编写CryptLang程序时，使用的函数变量名以“\_”开头，以便与代码生成后新添加的变量进行区分。

上述代码中，合约名为MedicalRecord即医疗记录，函数名为permit即授权（患者可以通过签名的方式，将自己的医疗记录的查询权限授权给医生），函数的三个变量为\_owner（患者的地址），\_hospital（患者曾就诊的医院），\_year（患者授权的记录年限）。

“@ECDSA with SHA3-256”表示，生成ECDSA签名的验证合约，合约中所有的哈希方案为keccak256。签名者的验证地址默认为第一个变量\_owner，后续的两个变量\_hospital, \_year为签名的内容信息。

“\_search”函数用于查找\_owner患者在\_hospital医院的\_year年内的医疗记录。

通过编译器自动生成的Solidity是一个可以直接部署在区块链上的签名验证与记录查询合约。

编译器会生成ECDSA签名库，以及原有合约名的MedicalRecord合约。此外，还会生成一个mapping类型的nonce，即为每个签名者单独存储一个随机数用于防止重放攻击。

编译器会在permit函数的参数列表中，多加入一个sig变量用于表示输入的签名。hash除了包括必须的签名内容（\_owner, \_hospital, \_year）之外还加入了nonce[\_owner]和address(this)，即随机数和合约本身的地址，用于防止重放攻击。之后会对恢复出的签名地址与签名者地址进行比较，若验证成功则正常调用\_search函数对用户医疗记录进行查找，若验证失败则报错。+

## 语法设计

CryptLang基于Antlr实现词法、语法分析、代码翻译与转义。下图展示了CryptLang的语法规则定义。

taskSymbol

: '@' ;

addrSymbol

: '#' ;

privateIdentifier

: privateSymbol? identifier ;

privateIdentifierList

: ( privateIdentifier? ',' )\* privateIdentifier? ;

hashMethod

: 'SHA3' | 'SHA2' | 'RIPEMD' ;

signatureMethod

: 'ECDSA' | 'RSA' | 'BLS';

commitmentMethod

: 'Pedersen' | 'Merkle' ;

proofMethod

: 'Groth16' | 'PLONK' ;

signatureStatement

: statementSymbol signatureMethod ('with' hashMethod)? '(' ('#' identifier)? ( ',' identifier)\* ')';

commitmentStatement

: statementSymbol commitmentMethod ('with' hashMethod)? '(' (identifier? ',')\* identifier? ')';

taskStatement

: (signatureStatement | commitmentStatement) ';' ;

otherStatement

: .+? ';' ;

图 53 CryptLang语法规范

上述代码为Cryptlang的g4代码，基于solidity进行扩充设计。

下面将逐条解释这些规则的意义：

taskSymbol：'@'：定义了任务符号为@。

addrSymbol：'#'：定义了地址符号为#。

privateIdentifier：privateSymbol? identifier：一个可选的私有符号（privateSymbol，未在您的示例中定义）后跟一个标识符（identifier）。

privateIdentifierList：( privateIdentifier? ',' )\* privateIdentifier?：这是一个以逗号分隔的私有标识符列表，其中每个私有标识符都是可选的，列表中可以没有私有标识符。

hashMethod：'SHA3' | 'SHA2' | 'RIPEMD'：定义了可用的哈希方法，包括SHA3, SHA2, 和RIPEMD。

signatureMethod：'ECDSA' | 'RSA' | 'BLS'：定义了签名方法，包括ECDSA, RSA, 和BLS。

commitmentMethod：'Pedersen' | 'Merkle'：定义了承诺方法，包括Pedersen和Merkle。

proofMethod：'Groth16' | 'PLONK'：定义了证明方法，包括Groth16和PLONK。

signatureStatement：statementSymbol signatureMethod ('with' hashMethod)? '(' ('#' identifier)? ( ',' identifier)\* ')'：定义了签名语句的结构，其中包括一个语句符号（statementSymbol），一个签名方法，可选地指定一个哈希方法，以及一个圆括号中的标识符列表，列表中的第一个标识符可以是以#为前缀的。

commitmentStatement：statementSymbol commitmentMethod ('with' hashMethod)? '(' (identifier? ',')\* identifier? ')'：定义了承诺语句的结构，与签名语句类似，但用于承诺方法。

taskStatement：(signatureStatement | commitmentStatement) ';'：定义了任务语句，可以是签名语句或承诺语句，以分号结束。

otherStatement：.+? ';'：匹配除了前面定义的语句外的任何其他语句，直到遇到分号结束。

更具体地来说，首先是两个特殊符号，“@”是Cryptlang语法的启始标识符，当用户使用此符号时，表示其准备启用Cryptlang的密码学功能。“#”符号是专门为Cryptlang中的零知识证明方法定制的符号，用户需要在零知识证明的隐私输入变量符号前加上“#”以便与公开输入进行区分。

接下来我们分别定义四个密码学方法：

哈希：SHA3-256即solidity中的keccak256，SHA2-256即solidity中的sha256，RIPEMD160即solidity中的ripemd160。在以太坊上，哈希函数被广泛应用于各种场景，包括数据完整性验证、智能合约地址生成、加密和安全性等方面。在以太坊中常用的哈希函数有SHA3-256(keccak256)，SHA2-256(sha256)和RIPEMD-160(ripemd160)。它们都可以由以太坊原生的密码学接口直接实现，keccak256可以通过调用以太坊的opcode，sha256和ripemd160可以通过调用以太坊内置的预编译合约来实现。

签名：分别为ECDSA的签名验证和BLS的签名验证。签名验证是另一个在以太坊上的常见应用，可用于身份认证，授权和访问控制，安全性验证等等。签名验证是以太坊上保障安全性和可信性的重要机制之一，通过在交易和消息中使用数字签名，可以确保只有拥有相应私钥的用户能够执行特定操作，并防止网络中的欺诈行为。以太坊中最常用的签名方案为ECDSA，因此以太坊的第一个智能合约就是secp256k1曲线上的ECDSA签名验证预编译合约。通过调用合约，可以恢复出签名者的公钥相关联的地址，从而验证签名的有效性。然而ECDSA不能进行签名聚合，BLS签名可以进行聚合。BLS签名聚合是一种基于pairing的签名聚合方案，它可以将多个签名聚合成一个签名，从而减少交易的大小和成本。然而使用BLS签名聚合时，需要运算G\_T上的乘法，目前以太坊并没有提供相应的预编译合约，但是有望在下一次升级中加入BLS-12-381曲线上的pairing预编译合约。

承诺：分别为Pedersen承诺和Merkle承诺。承诺是密码学中的一个重要概念，通过承诺可以将某个值隐藏起来，同时保证该值不会被篡改。承诺可以用于实现零知识证明，可验证计算，可验证随机函数等等。区块链上的常用承诺协议有Pederson承诺和Merkle承诺。Pederson承诺是一个基于离散对数的承诺方案，可以通过调用以太坊的modExp预编译合约实现。Merkle承诺是一个基于哈希函数的承诺方案，同样可以通过调用以太坊的预编译合约实现。

零知识证明：分别为Groth16证明和PLONK证明，均为zk-snark。零知识证明是密码学中的一个重要概念，它可以用于证明某个语句的真实性，同时不泄露任何关于该语句的信息。零知识证明可以用于实现隐私保护，可验证计算，可验证随机函数等等。区块链上的常用零知识证明协议有Groth16和Plonk。Groth16是一个基于pairing的零知识证明方案，可以通过调用以太坊的pairing相关预编译合约实现。Plonk是一个基于多项式的零知识证明方案，同样可以通过调用以太坊的预编译合约实现。

最后是Cryptlang具体的密码学任务表示方式，本DSL提供三个密码学任务，分别为签名验证，承诺验证和零知识证明验证，下面我们将分别介绍： 在所有任务开始前都需要输入“@”符号。

签名验证：首先输入具体的签名方法名（ECDSA或BLS），接着输入可选的哈希方法（SHA3-256，SHA2-256或RIPEMD160），最后以输入函数参数的方式，按顺序输入签名者的地址以及所有被签名的参数。

承诺验证：首先输入具体的承诺方法名（Pedersen或Merkle），接着输入可选的哈希方法（SHA3-256，SHA2-256或RIPEMD160），最后以输入函数参数的方式输入承诺值。

零知识证明验证：首先输入具体的证明方法名（Groth16或PLONK），接着输入可选的哈希方法（SHA3-256，SHA2-256或RIPEMD160），最后以输入函数参数的方式，按顺序输入公开输入和隐私输入。

### 签名语法设计

通过数字签名可以实现区块链上的访问控制，确保只有授权的用户能够访问特定的区块链资源或执行特定操作。这可以在民生领域中用于访问医疗记录等敏感信息，以下是通过签名实现区块链上授权医疗访问的基本步骤：

授权医疗访问概述：患者可以使用签名的方式向医疗专业人员授权访问特定的医疗数据，以便他们能够提供准确的诊断和治疗建议。这确保了医疗专业人员只能访问被授权的数据，保护了其他敏感信息的隐私。

数字签名生成： 当用户想要访问某个资源或执行某个操作时，他们使用自己的私钥生成数字签名。数字签名基于用户的身份（签名的地址）和要访问的资源信息（一年内在北京大学口腔医院的所有就诊数据），是用户身份的验证标志。

请求提交： 用户将访问请求以及生成的数字签名提交到区块链网络中。

验证和授权： 区块链网络中的节点接收用户的访问请求，使用相应的公钥对数字签名进行解密和验证。如果验证通过，智能合约会判断用户是否有访问权限，如果有，就执行相应的操作，否则拒绝访问请求。

Cryptlang帮助医院等医疗记录存储机构快速生成验证签名并提供医疗记录的合约，具体代码在“CryptLang代码结构”小节中CryptLang程序示例中。

### 承诺语法设计

利用承诺协议可以在区块链上实现安全、私密和可信的投票过程。以下是利用承诺协议进行链上投票的基本思想和步骤：

承诺协议概述： 承诺协议是一种加密技术，允许参与者在不泄露真实数据的情况下，证明某种情况下的数据或行为。在投票场景中，承诺协议可以用来确保投票者的投票是私密的，同时也能在需要时证明其投票的有效性。

投票过程： 假设有一个需要投票的议题，参与者可以按以下步骤进行投票：

投票生成： 每位投票者将自己的投票选择加密成一个承诺。这个承诺不会明确透露投票的具体内容，但确保了投票者不能后续否认自己的投票选择。

承诺提交： 投票者将生成的承诺提交到区块链上。

投票验证： 在投票截止后，可以使用承诺协议的技术解密承诺，验证每个投票者的投票是否与其生成的承诺相匹配。

Cryptlang帮助政府等选举机构快速生成验证承诺并进行投票统计的合约，具体代码如下图所示：

contract Vote {

function count(bytes32 \_leaf) public pure {

@Merkle with SHA2-256(\_leaf);

\_count(\_leaf);

} ... }

对于用户编写的Cryptlang合约，Vote为投票合约，count为投票统计函数，其中\_leaf为承诺的哈希原像，当通过“@Merkle with SHA2-256”所生成的Merkle承诺验证后，就可以保证哈希原像的真实性，因而可以调用\_count函数进行后续的投票统计操作。

对于自动生成的Solidity合约，编译器会自动生成一个mapping类型的nonce，即为每个\_leaf单独存储一个随机数用于防止重放攻击。count函数加入了root和proof参数，即用户还需输入merkle tree的根哈希和证明才能对特定的\_leaf进行验证。接下来是自动生成的Merkle承诺验证算法，其中包括了Merkle树的构建和验证过程。若验证成功，则调用\_count函数进行投票统计，否则报错。

### 零知识证明语法设计

医疗数据包含敏感的个人健康信息，同时需要与医疗专业人员和机构共享，以提供准确的医疗服务。利用零知识证明和区块链技术，可以实现以下医疗隐私保护的应用：

隐私保护的医疗数据共享： 零知识证明可以使患者能够将医疗数据安全地共享给医疗专业人员或研究机构，而不必透露具体的细节。患者可以证明某些健康信息的真实性，如某种疾病的确诊，而不需要透露整个病历。

Cryptlang帮助医院等医疗机构快速生成零知识证明验证并提供隐私医疗数据共享的合约，具体代码如下图所示：

contract MedicalData {

function share(uint256 \_n) public {

@Groth16 with "factor.zok"(#a, #b, \_n);

\_share(\_n);

} ... }

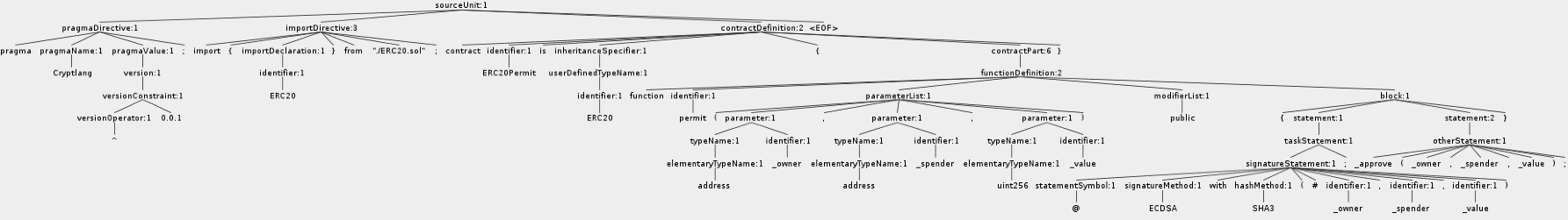
下图展示了“CryptLang代码结构”小节中CryptLang程序示例所对应的抽象语法树（AST）。

图 54 CryptLang程序示例所对应的抽象语法树（AST）

## 领域示范应用

### 领域场景

中国中医科学院中医药数据中心构建的中医可信联合循证分析平台（简称“联合分析平台”）通过智能合约来实现医药公司与医院科室之间的联合隐私计算任务。通过在项目中应用博雅正链智能合约领域特定语言族和CryptLang对现有智能合约开发流程进行改造，提高了业务人员和开发人员之间的需求沟通效率，并且能够直接生成智能合约代码，降低了业务系统研发周期和成本。通过快速编写签名验证，零知识证明等复杂密码学任务，解决了民生服务中医疗公司与医院科室针对病人数据的协同隐私计算问题。

### 示范应用

中国中医科学院中医药数据中心构建的中医可信联合循证分析平台（简称“联合分析平台”）支持面向横断面研究、队列研究、病例对照研究的智能合约和迁移学习模型，为跨医院、跨机构的中医循证联合分析应用提供数据授权、任务分发、共享贡献存证和基于合约的激励等联合应用服务，达到在数据不出医院的前提下，完成跨医疗机构的联合循证分析。

通过在博雅领域智能合约编辑软件v1.0图形界面编辑基本业务流程和民生领域智能合约编程语言CryptLang，工作人员能够快速实现中医药临床数据中患者人口学信息、就诊信息、检查检验信息等各类医疗业务信息的隐私保护逻辑，并生成智能合约代码，简化了智能合约的开发流程，降低了系统研发成本，有效支撑中医数据标准化管理、智能合约授权、计算模型发布、科研任务联合分析等应用服务。

针对这一场景，本项目基于智能合约领域特定语言族，设计并实现下述智能合约，利用智能合约领域特定语言族中的CryptLang模块，帮助医疗公司在链上发布数据收集任务，并于拥有对应数据的医院科室进行对接。通过CryptLang语言快速实现链上的签名验证，零知识证明等密码学任务，帮助公司和医院间的数据隐私计算。该成果可以较好地和现有系统集成，使用期间运行正常，达到了预期效果。

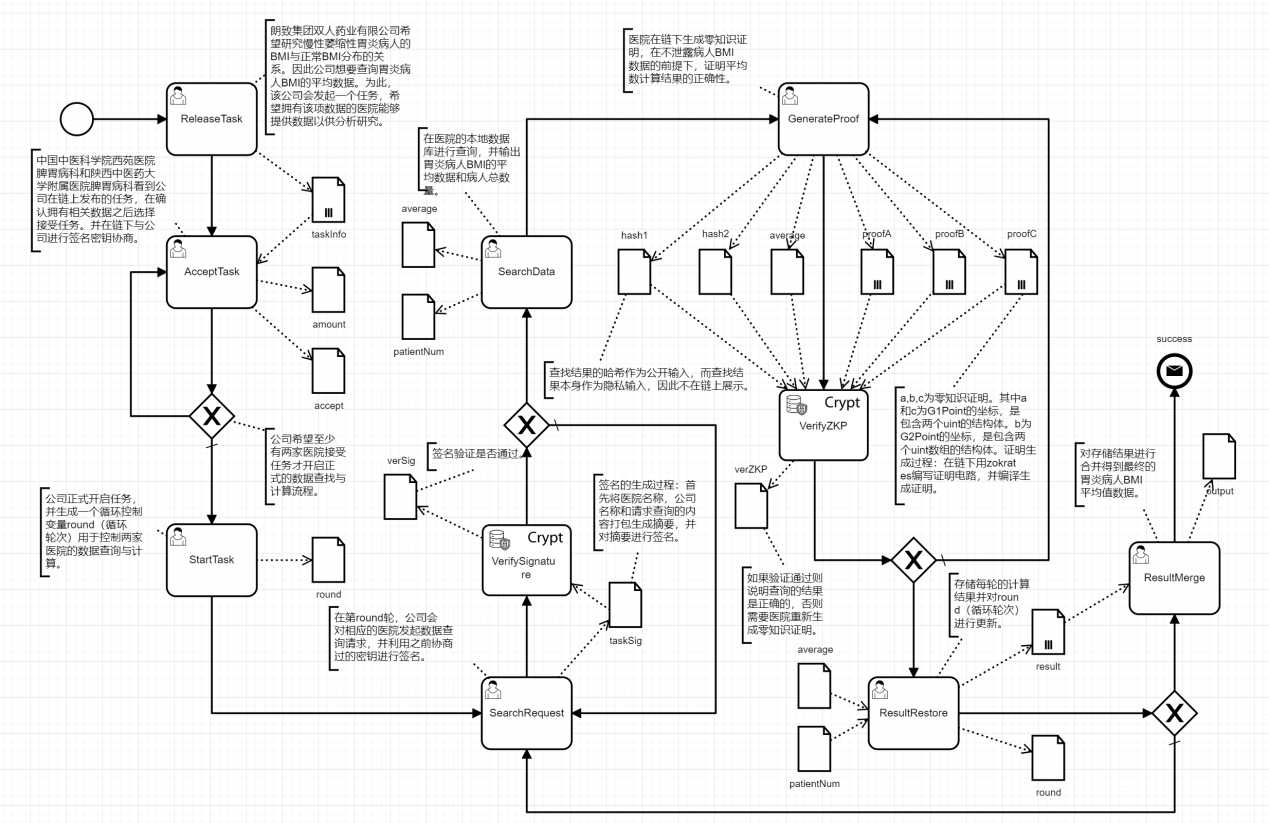


图 55 利用智能合约领域特定语言族编写的民生领域示范应用

# 技术实现

## 系统总体设计



图 56 技术实现总体设计图

### 系统功能模块概览

本系统的总体设计如图 56所示。图中以圆角矩形为形状的“BPMN业务流程建模”和“后端语义转换”是本系统的两大功能模块，分别对应着系统前端和系统后端的核心功能。其中BPMN业务流程建模模块代表着使用前端可视化界面绘制BPMN业务流程图的功能。该模块输出BPMN业务流程模型文件，以序列化的形式存储着用户绘制的BPMN流程图，通常为以bpmn为后缀的XML格式文件。后端语义转换模块代表着系统后端在收到来自前端的业务流程模型后，对XML格式文件进行解析，提取业务流程中涉及的基本BPMN组件、金融DSL组件、政务DSL组件、民生DSL组件，并根据其业务的逻辑关系和数据流传递来转换生成对应的智能合约文件。该模块输出的智能合约文件，是与用户绘制的业务流程模型相对应的智能合约，实现了由领域专家熟悉的BPMN业务流程建模到专业性较强、难度较大的链上智能合约的转换与生成。

### BPMN组件与DSL组件的关系

在两大功能模块中，部分子模块用蓝色和红色予以标识。其中标蓝色的模块代表BPMN 2.0规范中已有的或基于BPMN 2.0规范派生出的元素，例如基本BPMN组件、连接元素、数据对象、内部任务、外部任务等，以及根据BPMN规范中元素在业务流程中的意义，所构造的生成智能合约的转换规则、驱动智能合约按照业务流程运行的状态转移函数等；标红模块代表本项目自定义的领域特定语言模块，例如在基本BPMN组件上增加的领域特定语言扩展属性、独立于基本BPMN组件以外的领域特定语言专属组件、自定义领域特定语言（金融、政务、民生）的智能合约转换规则、生成的智能合约中可复用的逻辑和函数等。

由系统设计图可以看出，本系统中BPMN建模框架与领域特定语言组件之间为相互补充、协同运作的关系。一方面，BPMN作为业界常用的业务流程定义模型，在本系统中也起到基础框架的作用。领域专家可以使用自己熟悉的业务流程设计方法，使用BPMN框架完成金融、政务、民生领域的业务流程。其基本BPMN元素与链下的业务流程设计平台没有区别，因此本系统可以通过使用基本BPMN元素，为领域专家提供无感的链上智能合约开发过程。另一方面，领域特定语言组件为BPMN框架赋能，提供了为解决领域特定需求的兼具语法简洁性和领域应用表达能力的DSL。使得领域专家可以方便地使用DSL来完成领域特定需求，并通过本系统后端语义转换生成能满足上述需求的链上智能合约。同时，领域专家还可以将领域特定语言组件与其他基本BPMN组件相互串联和组合，以实现更复杂的业务逻辑、满足更丰富的业务场景。通过本系统业务流程建模功能模块设计的包括基本BPMN组件和自定义DSL组件的业务流程，可以经由后端语义转换功能模块生成智能合约，其兼具保持原有业务的流程和逻辑，以及支持自定义DSL在链上智能合约中的转换与表达，有效解决了通用智能合约编程语言中难以充分表达领域知识，和领域专家难以实现智能合约领域模型的准确构建与高效表达的痛点。

### 前端BPMN业务流程建模模块

前端BPMN业务流程建模模块基于Web前端技术进行设计和开发，采用Vue渐进式Web框架和bpmn-js库来实现，用于在Web界面上创建、设计、编辑BPMN业务流程图。前端以组件库的形式为领域专家提供灵活的BPMN组件配置，使领域专家可以通过可插拔的方式选取并配置业务流程所需要的BPMN组件。可选择的BPMN组件包括BPMN 2.0规范中的基本组件，包括任务、事件、网关、顺序流、数据对象、数据存储等。此外，本项目前端还支持对基本BPMN元素进行扩展，增加自定义DSL的扩展属性，如对基本BPMN任务增加访问控制权限的支持。使得符合领域特定需求的DSL可以为基本BPMN元素赋能，更好地发挥BPMN业务流程在解决领域实际问题中的作用。除了BPMN 2.0规范中的基本组件，前端还提供了金融、政务、民生各自DSL单独的组件。这些组件主要用于填写实现领域特定需求的DSL代码片段，通过设计DSL时兼具语法简洁性和领域应用表达能力的目标，实现领域专家对领域特定需求的高效表达与解决。BPMN组件库中的各个组件为并列关系，用户可以组合使用基本BPMN组件，和金融、政务、民生各自的DSL组件。组件之间的逻辑关系串联通过BPMN连接元素实现。连接元素包括普通顺序流、条件顺序流、默认顺序流等，定义了BPMN业务流程模型中的业务逻辑，即业务流在BPMN元素之间应以何种顺序进行执行。此外，本项目前端还支持了对业务数据流的定义。通过使用BPMN数据对象，用户可以在定义业务逻辑控制流以外定义业务的数据流，即业务数据在业务流程中所进行的处理。例如，BPMN任务的执行通常涉及数据的输入与输出，因此对任务的输入与输出数据在业务流程图上进行标注，有助于领域专家更清晰地表达整个业务过程涉及的对象。总而言之，本系统前端的设计除了支持基本BPMN元素的建模与编辑，还包括以模块化的形式增加自定义DSL功能组件，并通过BPMN连接元素与BPMN数据对象，共同定义了业务的逻辑控制流与数据流，为领域专家提供了准确表达业务模型的方法。

### 后端语义转换模块

后端语义转换模块用于转换BPMN业务流程模型为链上智能合约，其输入是来自﻿BPMN业务流程建模模块的XML格式业务流程，输出是GCL、Solidity、Move等语言的链上智能合约。语义转换模块中的转换规则和目标语言均以组件化、可配置的思路进行设计。其中转换规则包括基本BPMN元素到智能合约的转换规则，其根据BPMN连接元素和数据对象所确定的逻辑控制流和数据流，生成智能合约中与业务流程对应的智能合约代码。除基本BPMN元素的转换规则外，系统后端还配置了金融DSL、政务DSL、民生DSL各自的生成智能合约的转换规则，作用是根据自定义DSL的语法规则和语义模型生成智能合约代码。以上三种DSL的转换规则以模块化组件的形式配置在语义转换模块中，使得领域专家可以根据领域中新的需求，或其他领域的问题添加新的DSL转换规则，保证了本系统具有较强的领域通用性。此外，后端语义转换模块还支持其他目标语言的规则转换模块，例如可以将金融DSL生成Solidity的转换规则替换为生成GCL的转换规则，从而生成基于GCL语言的智能合约代码，解决了无法在异构区块链平台进行智能合约引擎可插拔替换的痛点。

### 后端生成的智能合约

本小节将对后端最终生成的智能合约总体设计进行说明。该智能合约是经由后端语义转换模块，将前端建模的业务流程模型根据BPMN元素转换规则而得到的，与前端业务流程逻辑对应的智能合约。该智能合约总体框架基于一个状态机来运行，状态机中的状态是执行各个不同类别的任务，包括执行基本BPMN内部任务、基本BPMN外部任务、金融DSL任务、政务DSL任务、民生DSL任务等。其中基本BPMN内部任务的执行均放在链上执行，如简单的链上脚本代码，网关判断等；基本BPMN外部任务需要和链下的外部对象进行交互，因此在生成的智能合约中，需要添加与外部对象交互的工具函数等组件。在执行BPMN外部任务时，也需要调用这些﻿与链下外部对象交互的组件；金融、政务、民生等领域的DSL任务，则需要在生成的智能合约中添加各自领域中所需要的工具函数等组件，同时在运行合约时，也需要调用这些自定义DSL的相关组件。

上述状态机中的各个状态，是在﻿基于BPMN业务逻辑的状态转移函数的驱动下运行的。根据前端建模的BPMN业务流程，后端语义转换模块生成状态转移函数，让状态机运行各个状态的顺序和逻辑与业务流程一致。对于每个前端建模的BPMN任务，都会在后端语义转换模块中转换为一个状态，状态转移函数根据前端BPMN任务的顺序和逻辑确定智能合约中各个状态的执行顺序。

## 前端技术选型

本项目前端采用Vue框架进行开发，结合ElementUI-Plus框架进行UI界面开发，确保整体UI呈现一致的风格，并利用Vite构建工具提高开发效率。本项目基于BPMN-JS库进行开发，让用户可以轻松地通过拖拽模块的方式来设计业务流程。为了避免用户设计的流程意外丢失，本项目充分利用了VueX来可靠地存储这些设计数据。同时，本项目还利用Axios与后端进行通信，将设计好的流程图转换为JSON格式并传输给后端。后端负责根据接收到的JSON代码进行解析和转换，最终生成Solidity语言的智能合约。

### Vue

Vue是一款友好、多用途、高性能的渐进式JavaScript框架，创建可维护性和可测试性更强的代码库，通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件。它采用了虚拟DOM技术，在内存中维护一个轻量级的虚拟DOM树。通过高效地计算虚拟DOM的差异，并最小化对实际DOM的操作，从而提高应用的性能，这种优化能够降低页面渲染的开销，提升用户体验。

本项目前端选用Vue框架进行开发，能够提高项目的可维护性和可测试性，加快开发效率，实现响应式的数据绑定，并以此来构建许多可复用的组件，大大减少了代码量，优化应用的性能。

### ElementUI-Plus

ElementUI-Plus是一个基于Vue的桌面端UI框架，它提供了丰富的Vue基础组件和工具，能够极大地加速本项目的前端开发过程。使用ElementUI-Plus框架，可以快速构建出符合项目需求的界面，节省大量的开发时间和工作量。此外，ElementUI-Plus的组件库设计精良，视觉风格简洁、一致，能够保持整个应用程序的视觉统一性，提升用户体验和品牌形象。通过使用ElementUI-Plus的组件，可以避免从头编写复杂的UI组件，减少了开发难度和错误，提高了开发效率。同时，ElementUI-Plus提供了丰富的配置选项和样式类，方便开发者根据项目需求进行定制和扩展。

本项目选用ElementUI-Plus作为UI框架能够加速开发过程，保证界面的一致性和用户体验，提高开发效率和质量。

### BPMN-JS

BPMN-JS是一个功能强大的JavaScript库，专门用于处理BPMN（Business Process Model and Notation）文件。BPMN是一种业务流程建模标准，用于描述业务过程和工作流程，以图形化的方式呈现各种业务操作、流程和决策。BPMN-JS的目标是简化BPMN图表的创建、编辑、展示和导入/导出，从而使其更容易集成到各种应用程序中。

本项目选用BPMN-JS库来实现可视化业务流程建模，它提供了标准化的业务流程建模功能，符合行业标准，还有着丰富的生态系统和活跃的社区支持，适用于本项目的可视化业务流程建模的需求，提供了可靠的解决方案。

### Vuex

Vuex是一个基于Vue的轻量级状态管理库，它提供了简洁、直观的API设计，能够帮助本项目更好地管理和组织应用程序的状态。使用Vuex，能够轻松创建和管理多个独立的store实例，每个store都有自己的状态和操作方法，使得代码更加模块化、可维护。Vuex还支持类型推断和TypeScript，能够带来更好的代码提示和类型检查，提高代码的可靠性和可测试性。此外，Vuex还提供了插件机制，可以方便地扩展其功能，集成其他库或实现特定需求。

本项目选用Vuex作为状态管理库，能够更好地组织和管理应用程序的状态，提升开发效率，减少代码冗余，同时提供更好的可维护性和可测试性。

### Axios

Axios是一款简洁、易用且功能强大的JavaScript HTTP客户端。它具有直观的API设计，支持各种类型的HTTP请求，并提供了Promise API，使得处理请求和响应变得简单。axios在浏览器和Node.js环境中都可以使用，具备跨平台兼容性，方便开发者在不同环境中使用相同的代码。此外，axios还提供了请求拦截器和响应拦截器等高级功能，可以进行全局的拦截和处理。它拥有强大的功能和灵活的配置选项，能够满足各种复杂的业务需求。

本项目选用axios作为HTTP请求库能够简化开发过程，提高代码的可读性和可维护性。

### Vite

Vite是一款快速、简单且现代化的前端构建工具。它具有快速启动和热模块替换（HMR）功能，加速了开发迭代的速度，提高了开发效率。Vite采用了原生ES模块的方式处理依赖关系，无需打包，使得开发环境轻量且响应更快。Vite与主流前端工具和框架的兼容性好，可以与Vue等无缝集成，发挥它们的优势。本项目选用Vite工具来构建前端项目，使开发过程中可以支持热模块替换，方便了开发与测试过程。

## 前端关键技术实现

### 添加自定义BPMN模块

本项目对标准BPMN规范进行了扩展以支持金融、政务、民生等领域特定的DSL。其中每个DSL以一个单独BPMN元素呈现，与任务平级。例如，Reglang模块起到的作用是在业务流程中，根据一定的监管规则对业务流程进行审查。因此，自定义的DSL模块与标准BPMN中的任务、网关等元素处于同一级别。

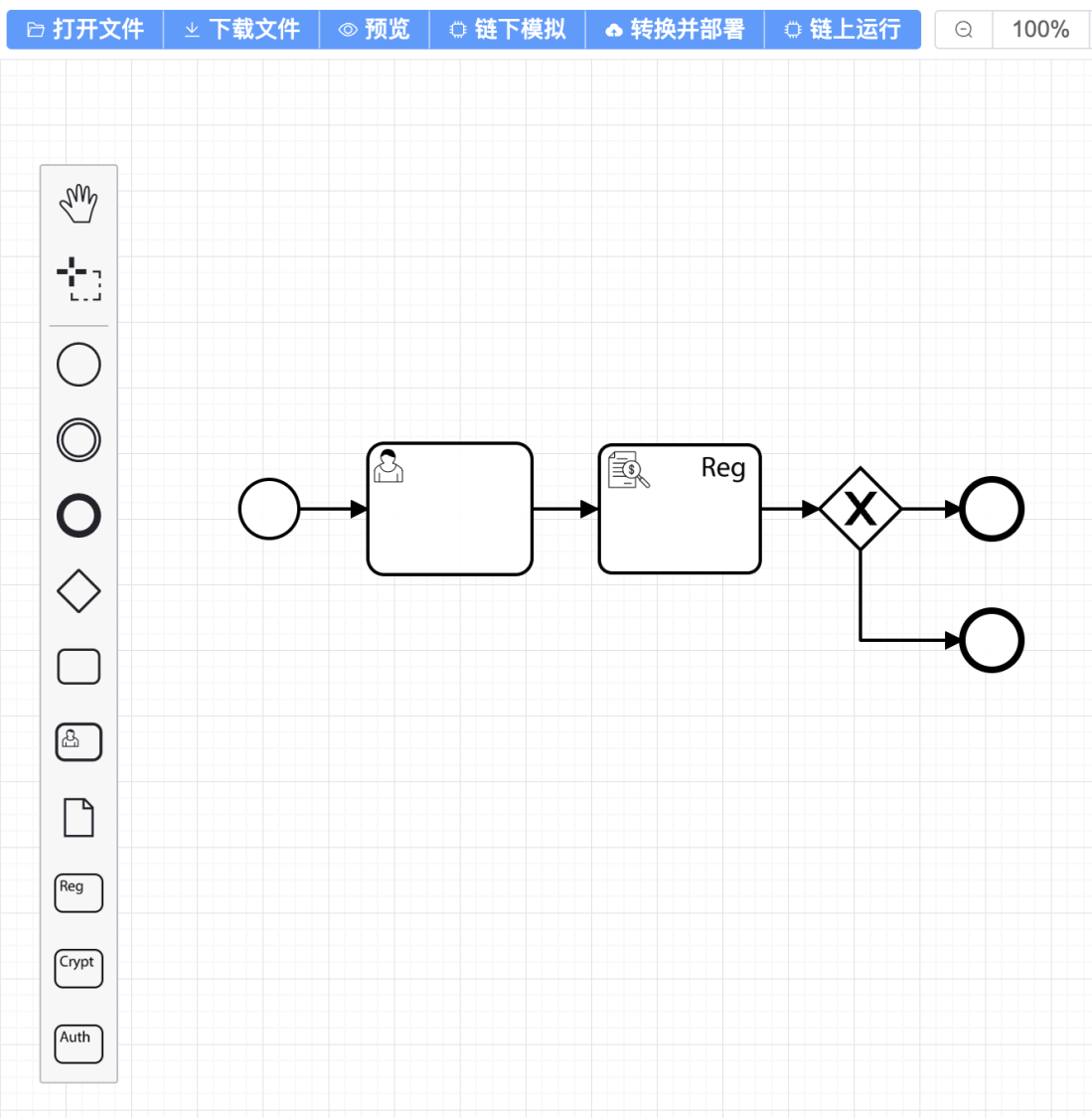


图 57 领域特定语言族中模块示意图

要自定义模块并将其集成到BPMN-JS中，需要执行以下步骤：在定义所有BPMN元素模块的json配置文件中加入自定义模块的描述信息，需要说明该模块名称(name)、继承的BPMN-JS中的基础模块(superClass)。例如，在如下代码片段中，本项目新增了RegLang、CryptLang、PolitikLang三种扩展的DSL模块，其继承的BPMN基础模块为普通任务模块（bpmn:Task），所在的大类别为Flowable（即使用flowable:RegLang来标识扩展的RegLang模块）。

1. {
2. "name": "Flowable",
3. "uri": "http://flowable.org/bpmn",
4. "prefix": "flowable",
5. "xml": {
6. "tagAlias": "lowerCase"
7. },
8. "associations": [],
9. "types": [
10. {
11. "name": "RegLang",
12. "superClass": [
13. "bpmn:Task"
14. ]
15. },
16. {
17. "name": "CryptLang",
18. "superClass": [
19. "bpmn:Task"
20. ]
21. },
22. {
23. "name": "PolitikLang",
24. "superClass": [
25. "bpmn:Task"
26. ]
27. },
28. ……
29. ]
30. }

图 58 集成了自定义BPMN模块的json文件

随后需要在BPMN-JS的palette模块（左侧显示所有可选元素的面板）中加入自定义模块。在渲染左侧面板的JS文件中，加入自定义模块放入左侧面板分组(group)、渲染时的样式(className)、显示模块名(title)以及开始拖动事件(dragstart)和点击事件(click)触发的方法。

1. 'create.reglang': {
2. group: 'model',
3. className: 'icon-custom reglang',
4. title: translate('创建一个RegLang任务节点'),
5. action: {
6. dragstart: createRegLang,
7. click: createRegLang
8. }
9. },
10. function createRegLang(event) {
11. const shape = elementFactory.createShape({
12. type: 'flowable:RegLang',
13. x: 0,
14. y: 0,
15. });
16. create.start(event, shape);
17. }

图 59 用于渲染自定义模块的代码片段

在负责BPMN图标绘制的JS文件中，添加自定义模块的图标，配置与图标相关的尺寸、默认位置等信息。同时需要重写父类中的原型函数，使BPMN-JS框架负责绘制图标的方法能在绘制基础BPMN元素的基础上，识别本项目新添加的三种自定义模块。其中配置自定义模块的方法如下：

1. import reglangImage from '@/assets/dollar.png';
2. import politiklangImage from '@/assets/politik.png';
3. import CryptLangImage from '@/assets/crypto.png';
4. const customConfig = { *// 自定义元素的配置*
5. 'flowable:RegLang': {
6. 'url': reglangImage,
7. 'attr': {x: 0, y: 0, width: 48, height: 48}
8. },
9. 'flowable:PolitikLang': {
10. 'url': politiklangImage,
11. 'attr': {x: 0, y: 0, width: 48, height: 48}
12. },
13. 'flowable:CryptLang': {
14. 'url': CryptLangImage,
15. 'attr': {x: 0, y: 0, width: 48, height: 48}
16. }
17. }
18. const customElements = ['flowable:RegLang', 'flowable:PolitikLang', 'flowable:CryptLang']

图 60 配置自定义模块显示效果的代码片段

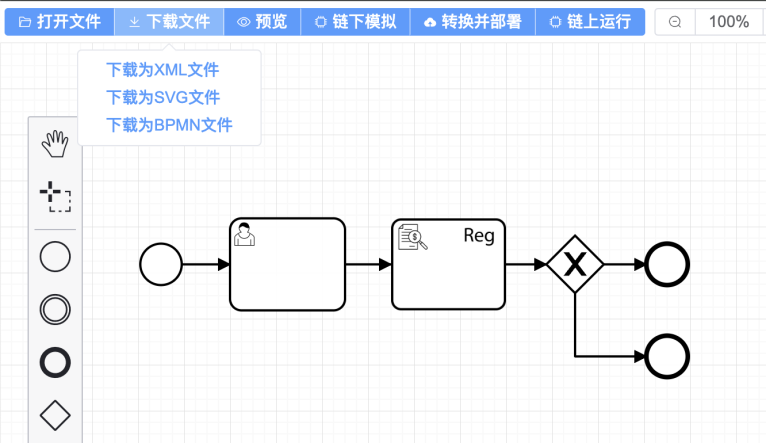
重写负责绘制BPMN图标的父类原型函数的方法如下：

1. const F = function () {
2. }; *//利用空对象作为中介；*
3. F.prototype = *BpmnRenderer*.prototype; *//将父类的原型赋值给空对象F；*
4. F.prototype.drawShape = function (parentGfx, element) {
5. const type = element.type *// 获取到类型*
6. if (customElements.includes(type)) {
7. const {url, attr} = customConfig[type]
8. const customIcon = *svgCreate*('image', { *//创建一个image*
9. ...attr,
10. href: url
11. })
12. element['width'] = attr.width
13. element['height'] = attr.height
14. *svgAppend*(parentGfx, customIcon)
15. return customIcon
16. }
17. *//复制父类中drawShape的实现*
18. var h = this.handlers[type];
19. return h(parentGfx, element);
20. }
21. *CustomRenderer*.$inject = ['eventBus', 'styles', 'pathMap', 'canvas', 'textRenderer'];
22. *CustomRenderer*.prototype = new F(); *//将 F的实例赋值给子类；*
23. *CustomRenderer*.prototype.constructor = *CustomRenderer*; *// 修复子类CustomRenderer的构造器指向，防止原型链的混乱；*

图 61 绘制BPMN图标的父类原型函数代码片段

### 下载流程图

设计好的流程图可转换为XML、SVG以及BPMN文件下载，转换主要依赖BPMN的save方法，当不同按钮点击调用downloadProcess函数时会传入不同的type参数，以此来区分需要转换的文件，从而调用BPMN中不同类型的save方法完成转换。其在前端中的界面和实现方法如下：



1. async downloadProcess(type, name) {
2. try {
3. const \_this = this;
4. *// 按需要类型创建文件并下载*
5. if (type === "xml" || type === "bpmn") {
6. const {err, xml} = await this.bpmnModeler.saveXML();
7. *// 读取异常时抛出异常*
8. if (err) {
9. console.error(`[Process Designer Warn ]: ${err.message || err}`);
10. }
11. let {href, filename} = \_this.setEncoded(type, name, xml);
12. this.downloadFunc(href, filename);
13. } else {
14. const {err, svg} = await this.bpmnModeler.saveSVG();
15. *// 读取异常时抛出异常*
16. if (err) {
17. return console.error(err);
18. }
19. let {href, filename} = \_this.setEncoded("SVG", name, svg);
20. this.downloadFunc(href, filename);
21. }
22. } catch (e) {
23. console.error(`[Process Designer Warn ]: ${e.message || e}`);
24. }
25. },

图 62 按文件类型分别下载流程图持久化文件

具体下载文件则依靠downloadFunc函数，该函数通过创建<a>标签，改变其href属性，并调用click方法实现下载。

1. *// 文件下载方法*
2. downloadFunc(href, filename) {
3. if (href && filename) {
4. let a = document.createElement("a");
5. a.download = filename; *//指定下载的文件名*
6. a.href = href; *// URL对象*
7. a.click(); *// 模拟点击*
8. URL.revokeObjectURL(a.href); *// 释放URL 对象*
9. }
10. },

图 63 下载流程图持久化文件的代码片段

### 读取流程图

符合BPMN规范的XML文件可以通过本项目前端转换为图形化的业务流程流程，其主要依赖于BPMN-JS的importXML方法，当用户点击打开文件并选择相应的XML文件后则会执行importLocalFile方法，该方法通过FileReader读取选择文件的文本内容获取到XML，随后将XML作为参数调用createNewDiagram方法，该方法会验证传入的XML是否为空，如果不为空则会调用BPMN-JS的importXML方法并将xml传入，此时则完成了读取流程图的操作。完成以上操作的关键实现函数如下：

1. *// 加载本地文件*
2. importLocalFile() {
3. const that = this;
4. const file = this.$refs.refFile.files[0];
5. const reader = new *FileReader*();
6. reader.readAsText(file);
7. reader.onload = function () {
8. let xmlStr = this.result;
9. that.createNewDiagram(xmlStr);
10. };
11. },
12. */\* 创建新的流程图 \*/*
13. async createNewDiagram(xml) {
14. *// 将字符串转换成图显示出来*
15. let newId = this.processId || `Process\_${new Date().getTime()}`;
16. let newName = this.processName || `Process\_${new Date().getTime()}`;
17. let xmlString = xml || *DefaultEmptyXML*(newId, newName, this.prefix);
18. try {
19. let {warnings} = await this.bpmnModeler.importXML(xmlString);
20. if (warnings && warnings.length) {
21. warnings.forEach(warn => console.warn(warn));
22. }
23. } catch (e) {
24. console.error(`[Process Designer Warn]: ${e?.message || e}`);
25. }
26. },

图 64 读取流程图持久化文件的代码片段

### 链下模拟运行

完成流程图后，可以通过点击“链下模拟”按钮开启模拟模式，模拟模式可将代码的运行情况、流程走向可视化的呈现出来。需要注意的是，链下模拟功能全部是在浏览器前端依据BPMN流程本身的逻辑而模拟的，与后端实际链上执行状态无关。如下图所示，启动模拟模块后，一个代表流程运行状态的令牌会从开始事件出发，按照业务流程顺序经过各个BPMN元素。若经过的是任务模块，则会在页面左下方弹出该任务名以进行提示；若即将经过的是排他网关，则允许用户在页面上选择希望执行的输出顺序流，随后令牌将沿着用户选择的输出顺序流继续执行。注意到，排他网关的输出是用户在模拟时指定的，而并不一定与程序运行的实际状况一致。这是因为链下模拟功能是基于前端而运行的，这部分并不与后端区块链实际运行情况相交互，因此该功能更多用于业务流程开发人员在设计业务流程时模拟使用，以测试业务流程的预期运行情况。

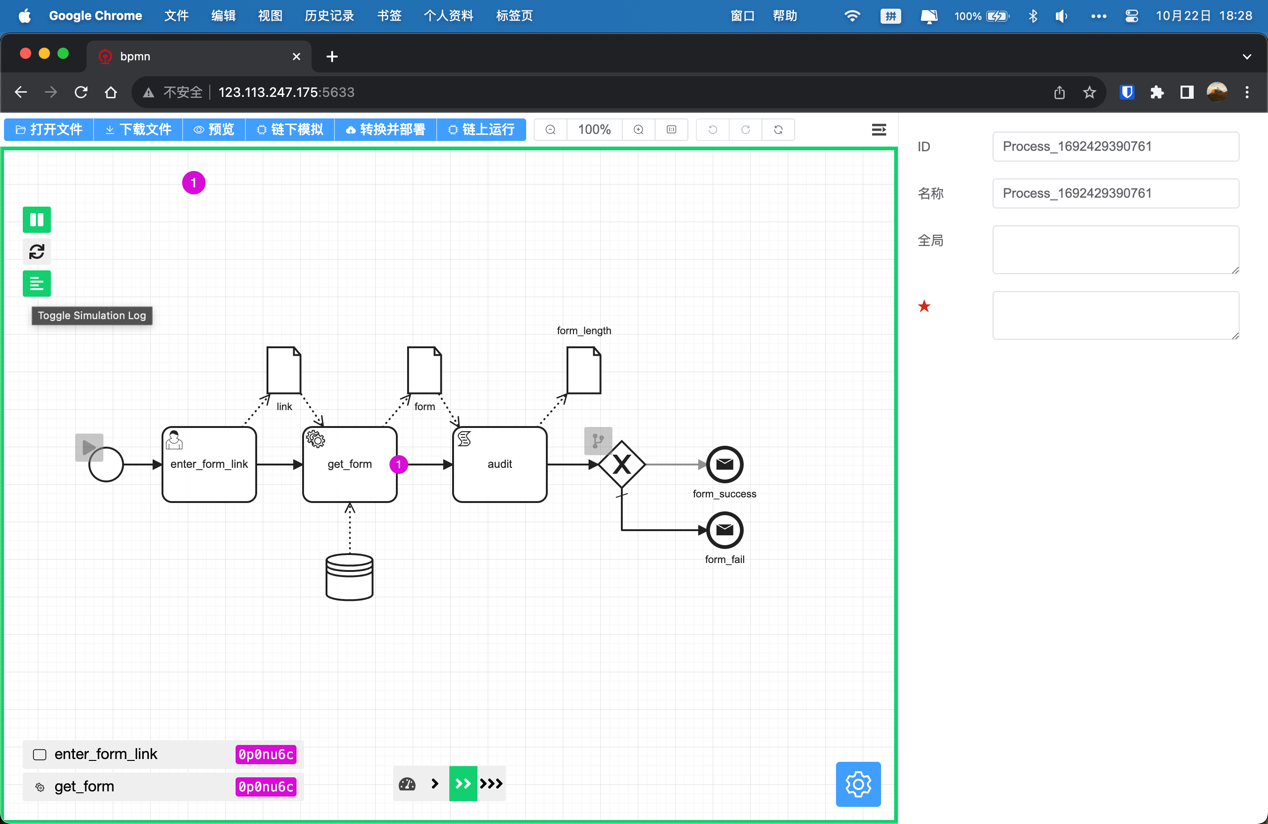


图 65 链下模拟运行界面示意图

模拟主要依赖于BPMN-JS的Simulation扩展插件，在BPMN-JS初始化时，通过向BPMN-JS添加该扩展插件就可使用该功能。

1. additionalModules() {
2. const Modules = [];
3. *// 模拟流转模块*
4. if (this.simulation) {
5. Modules.push(SimulationSupportModule)
6. Modules.push(tokenSimulation);
7. }
8. ……
9. },
10. initBpmnModeler() {
11. if (this.bpmnModeler)
12. return;
13. this.bpmnModeler = new *BpmnModeler*({
14. *// 绑定画布到dom上*
15. container: this.$refs["bpmn-canvas"],
16. *// 使用键盘快捷键*
17. keyboard: this.keyboard ? {bindTo: document} : null,
18. additionalModules: this.additionalModules,
19. moddleExtensions: this.moddleExtensions,
20. ...this.options
21. });
22. this.$emit("init-finished", this.bpmnModeler);
23. this.initModelListeners();
24. },

图 66 初始化流程模拟的代码片段

当用户点击“模拟”按钮时则会调用processSimualtion方法，该方法则会调用simulation模块的toggleMode方法开启模拟模式，此时该按钮文本会变换为”退出模拟”；再次点击”退出模拟”按钮同样会调用processSimualtion方法，并关闭模拟模式。

BPMN的模拟运行是基于bpmn-js-token-simulation仓库进行开发的（https://github.com/bpmn-io/bpmn-js-token-simulation），该仓库以动态、可交互的方式展现BPMN流程，与静态的BPMN流程图相比更能反映流程动态运行的本质。

### 链上运行

本项目对前端业务流程可视化建模与后端BPMN转化生成Solidity合约、运行合约进行了集成。后端除了支持从BPMN流程图生成Solidity语言的合约以外，还连通了一条测试链用于部署和运行生成的合约代码。因此，链上运行功能可以调用后端区块链平台提供的接口，对由BPMN生成的Solidity合约代码进行部署，并调用合约中的接口以实现业务流程的链上运行。链上运行时，外部任务通过抛出事件的方式将事件写入区块链运行时的日志中，链下监听程序对区块链日志进行监听并对抛出的事件进行解析，提取该外部任务所需要的外部输入（如用户任务）或所需要的外部服务（如服务任务），最后返回给前端进行交互。前端得到外部任务所需的数据后，再调用后端接口以调用链上合约，将获得到的数据以参数的形式传递上链，从而完成链上链下的交互过程。

前端实现链上运行功能的发送端的核心代码如下所示。链上运行模块与链下模拟模块共用一套前端界面，即bpmn-js-token-simulation库，区别在于链下模拟的令牌运行状态由用户自行调整前端界面来确定，而链上运行的令牌运行状态由后端实时发来的链上运行时事件来确定。代码清单第4行即开启令牌模拟界面。第8-19行规定发往后端的包的字段内容，其中重要的字段包括function\_name和args，定义了待调用的合约函数名和参数。来自前端的数据即通过这种方式发送至后端链上。

1. async proposeOneTx(isStart, proposer, args) {
2. const simulationSupport = this.bpmnModeler.get('simulationSupport')
3. const elementRegistry = this.bpmnModeler.get('elementRegistry')
4. simulationSupport.toggleSimulation(true)
5. let data
6. if (isStart) {
7. data = {
8. msgtype: Msg2Id.StartRun,
9. proposer: proposer,
10. args: [],
11. function\_name: 'startExecution'
12. }
13. } else {
14. data = {
15. msgtype: Msg2Id.StartRun,
16. proposer: proposer,
17. args: args,
18. function\_name: this.chainInteractName
19. }
20. }
21. const firstStopEvent = await *postBackend*(data)
22. }

图 67 链上运行过程的前端代码片段

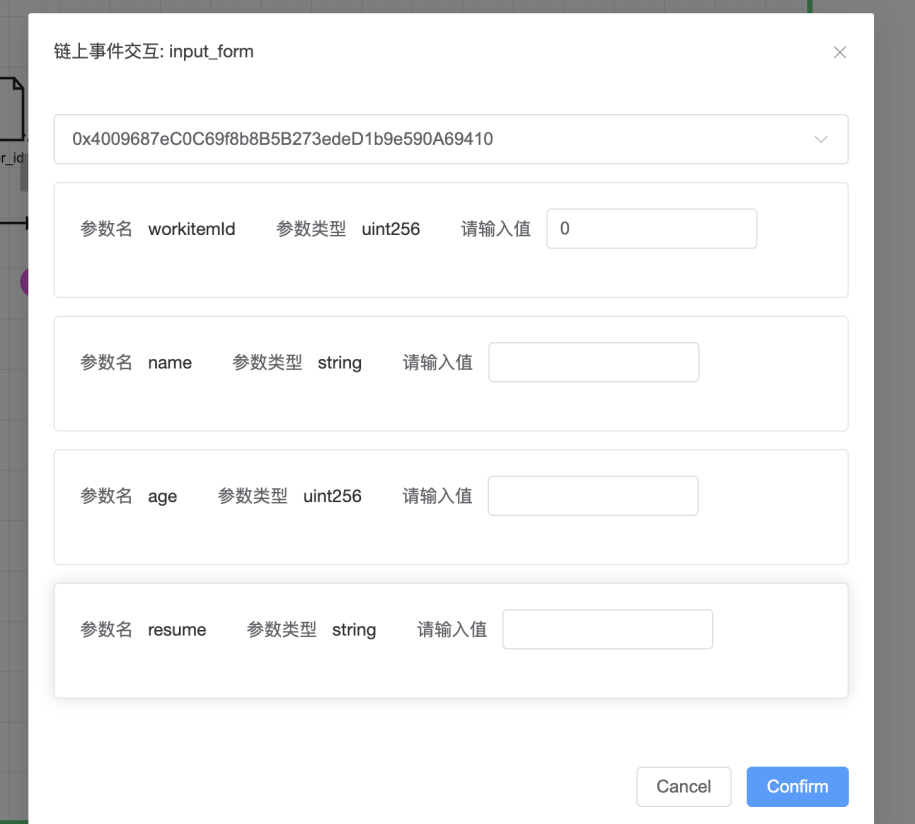


图 68 链上运行界面示意图

如上图所示，业务流程使用者在输入用户任务所需要的数据后，在前端界面点击右下角确认键，即等效于调用了后端合约中对应任务的函数（接口）。

当前端调用完后端用户任务的回调函数后，前端进入等待状态，等待后端返回执行完这个用户任务之后下一个需要前端（外部）进行交互的外部任务。核心代码如下所示：

1. const firstStopEvent = await *postBackend*(data)
2. ElMessage.success('运行成功')
3. const emittedEvents = firstStopEvent[0]
4. const lastTask = emittedEvents['execution\_progress'][emittedEvents['execution\_progress'].length - 1]['msg']
5. const lastType = elementRegistry.get(lastTask).type

图 69 用户任务与链下对象交互的代码片段

前端根据链上抛出的事件类别来执行不同的用户交互逻辑：若是由于执行到了需要与外部对象进行交互的外部任务（以用户任务为例），则前端弹出可交互式的输入窗口，待业务流程使用者输入以进行交互。待用户确认输入之后，前端再次发送以用户输入为参数的数据包至后端，重复本节开始的流程。核心代码如下所示：

1. this.chainInteractInput.splice(0)
2. for (const key in emittedEvents) {
3. if (key.endsWith('required')) {
4. this.chainInteractArgs = [...emittedEvents[key]]
5. }
6. if (key.endsWith('Requested')) {
7. this.interactProvidedArgs = [...emittedEvents[key]]
8. this.chainInteractInput.push(emittedEvents[key][0].index)
9. }
10. }
11. this.chainInteractName = elementRegistry.get(lastTask).businessObject.name
12. this.chainInteractVisible = true

图 70 用户任务与后端区块链交互的代码片段

若是由于执行到了结束事件，则前端抛出提示信息说明流程已运行结束。若结束事件类型为消息结束事件，则还会输出后端发送给前端的消息。核心代码如下所示：

1. if (lastType === 'bpmn:EndEvent') {
2. await simulationSupport.elementEnter(lastTask)
3. for (const key in emittedEvents) {
4. if (key.startsWith(lastTask)) {
5. const val = emittedEvents[key][0]['messageText']
6. ElMessage.success({
7. dangerouslyUseHTMLString: true,
8. message: `结束事件的消息是：${val} <br><br>流程已结束`
9. })
10. return
11. }
12. }
13. ElMessage.success('流程已结束')
14. return
15. }

图 71 前端执行结束事件的代码片段

## 后端关键技术实现

### 概览

本项目的总体目标是使用户通过前端图形化流程编辑界面创建业务流程，前端将用户创建的图形化业务流程生成为可序列化的JSON文件并传输给后端，最终后端解析此JSON文件并生成于业务流程对应的Solidity代码。因此，本项目后端的核心是完成从BPMN业务流程图到Solidity代码的转换过程，可以类比为一个BPMN到Solidity的编译器。输入的BPMN流程可包含用户任务、服务任务、脚本任务（附带Solidity脚本）、网关（排他、并行）、事件（默认、消息)、数据对象等基础BPMN元素，以及本项目支持的领域特定语言（DSL）所对应的扩展BPMN元素。输出则是能包括BPMN基础元素所组成的业务流程，以及本项目DSL所扩展的业务流程的Solidity代码。

本项目还支持对链上运行状态实时地进行可视化展示，将后端区块链平台上运行的合约状态反馈在前端BPMN流程图上。因此，本项目后端的另一个重要模块是维护前后端的交互信道，保障后端将合约运行状态返回给前端，并接收来自前端的外部任务输入请求。

本章节将从以下三个方面对项目后端进行介绍。首先对后端系统设计进行概述，介绍后端主要模块和程序执行流程。随后详细介绍将BPMN流程编译生成Solidity智能合约的过程，包括变量的处理方式以及如何通过用户任务和服务任务与外部资源交互，如何在生成的智能合约中编译控制流（顺序流、网关）和事件等内容。最后，本节将介绍前后端交互模块如何发出链上事件以及如何接收链下外部任务的输入。

### 系统后端设计概述

本项目后端基于Flask框架进行开发。Flask是一种基于Python语言编写的轻量级Web应用程序框架，用于快速开发Web应用程序和API。Flask提供了Web开发基本的工具和组件，并允许开发者自由选择和整合其他扩展使其高度可定制。同时由于其轻量级特性，Flask在性能方面表现出色，适用于处理大量并发请求。Flask与本项目后端开发的需求契合，即接收来自前端的BPMN业务流程请求，并调用后端代码编译生成Solidity文件，适当返回编译过程中生成的内容和信息给前端，在前后端交互方面不存在复杂的逻辑，因此采用Flask技术栈来进行后端开发是一个合适的选择。

本项目后端主要包括以下几个模块：

* + 主模块FlaskBackend用于维护一个基于Flask框架的Web服务，处理来自前端的GET和POST请求，并将请求数据传递给RequestHandler模块中的相应函数进行处理，随后将结果返回前端。
  + RequestHandler模块根据请求中的类别代码交由后端的不同模块来处理。本项目主要用到的类别代码包括：MainCovert，用于表明该请求的载荷是BPMN流程图所导出的JSON文件，需要由MainConverter模块转换生成为Solidity代码；StartRun，需要由ChainHandler模块调用合约上的函数以继续执行流程；GetAllAddrs，通过ChainHandler模块获取链上所有可用的账户地址。
  + MainCovert负责对由BPMN流程图所解析的JSON文件进行转换，生成与BPMN流程逻辑相对应的Solidity合约代码，即本章节后续重点介绍的部分。MainCovert完成转换后，将生成的Solidity代码通过RequestHandler模块返回给前端，完成前后端交互流程。
  + ChainHandler负责与后端运行地区块链平台进行交互。其主要通过Web3py开源Python库来实现区块链与链下Flask后端的交互，包括调用合约函数、监听链上事件等操作。

### 生成的合约框架

对于一个常规的BPMN流程（如1.5小节中的示例），MainConverter将会生成两个合约以实现对BPMN流程的链上模拟过程，这两个合约分别是主流程合约和Worklist合约。主流程合约用于模拟BPMN业务流程，包括模拟控制流和模拟数据流这两部分；Worklist合约用于实现链上流程与外部对象的交互，以实现链上运行的外部任务（用户任务、服务任务）需要与链下外部对象（如用户输入、数据库服务、网络服务）进行交互的要求。主流程合约和Worklist合约的框架如下所示：

1. pragma solidity ^0.8.0;
2. contract Process\_FormAudit\_Contract {
3. uint public marking = 2;
4. uint public startedActivities = 0;
5. address internal \_worklist = address(0);
6. event execution\_progress(string msg);
7. ...
8. string \_link;
9. ...
10. constructor (address worklist) { ... }
11. function startExecution() public { ... }
12. function enter\_form\_link\_complete(uint elementIndex, string memory link) external {…}
13. ...
14. function step(uint tmpMarking, uint tmpStartedActivities) internal { ... }
15. }
16. contract Process\_FormAudit\_Worklist {
17. ...
18. event enter\_form\_link\_Requested(uint index);
19. function enter\_form\_link\_start(uint elementIndex) external { ... }
20. function enter\_form\_link(uint workitemId, string memory link) external { ... }
21. }

图 72 主流程合约总体代码框架

主流程合约以状态机的形式驱动，在图 72所示代码第4、5行的两个变量共同控制状态机的运行状态。第18行step函数定义了该状态机的状态转移方程，根据第4、5行变量的状态来确定将要执行的模拟某个任务的一系列函数。对于每一个外部任务（如用户任务），MainCoverter模块都会生成三个函数来模拟该外部任务的行为，其中包括一个在主流程合约中，后缀为complete的函数（形式为<element name>\_complete）；和两个在worklist合约中，后缀为start的函数（形式为<element name>\_start）和以该任务为名称（形式为<element name>）的函数。worklist合约中的<element name>\_start函数用于触发代表该外部任务即将执行的事件，以使得外部对象能够捕捉到该事件并触发相应的链下操作；worklist合约中的<element name>函数是链下外部对象可调用的接口，用于外部任务执行完毕后将所得到的数据返回至链上。主流程合约中的<element name>\_complete是只能由worklist合约调用，其作用是在外部任务执行结束后，标记该任务的执行状态为完成并迁移至下一个状态。对于内部任务，MainCoverter模块会在状态转移函数step中生成对应的逻辑操作，整个过程放在链上合约的step函数中完成，不涉及与链下外部对象的交互。

除了上段所述的模拟BPMN任务的一系列函数，以上两个合约中其他的部分定义了区块链上事件。主流程合约中的execution\_progress事件为表示当前状态机运行状态的事件，当状态转移函数step被调用时，该事件将会被触发，以确保前端界面中显示合约运行状态的功能保持同步。worklist合约中以requested为结尾的事件用于表示状态机的运行状态已经到达了该外部任务，用于触发外部对象进一步处理该任务。

### 控制流转换过程

MainConverter模块对BPMN业务流程的转换主要分为控制流和数据流这两部分。控制流代表各个BPMN元素（包括任务、网关，以及连接这些元素的顺序流）和这些BPMN元素的先后顺序和逻辑关系；数据流代表运行该业务流程时，业务流程中涉及到的数据的定义、存储、传递方式。本小节将重点讨论控制流由BPMN流程到Solidity合约代码的转换过程，而在下一小节中重点讨论BPMN流程中数据的传递是如何转换至Solidity合约代码中的。

本项目支持的可转换为Solidity合约的BPMN元素主要可以分为以下两类：

**外部元素：**涉及外部资源的元素，即用户任务、接收任务、消息捕获事件和服务任务。

**内部元素：**所有除外部元素以外的其他元素，例如脚本任务、网关（排他、并行）以及开始事件、中间事件和结束事件（默认、消息）。

为将BPMN业务流程生成Solidity合约，本项目为BPMN元素规定了生命周期，分为外部元素生命周期和内部元素生命周期两种。在BPMN业务流程中，当代表流程执行状态的令牌出现在BPMN元素输入的边上时，该元素就会被启动执行（排他网关只需出现在一个输入边即可启动）。已启动的元素会使用掉输入一侧的令牌，并在输出一侧上生成新的令牌，从宏观上看即体现为该令牌“经过”了BPMN元素。相对而言，外部元素的生命周期相对复杂，其执行需要依赖于用户或外部服务的输入，通过向链上发送交易调用智能合约来实现。内部元素的生命周期相对简单的，由于没有外部交互，内部元素一旦在前一个元素执行完毕后就会开始执行。下文将分别介绍外部元素和内部元素的生命周期及其转换过程。

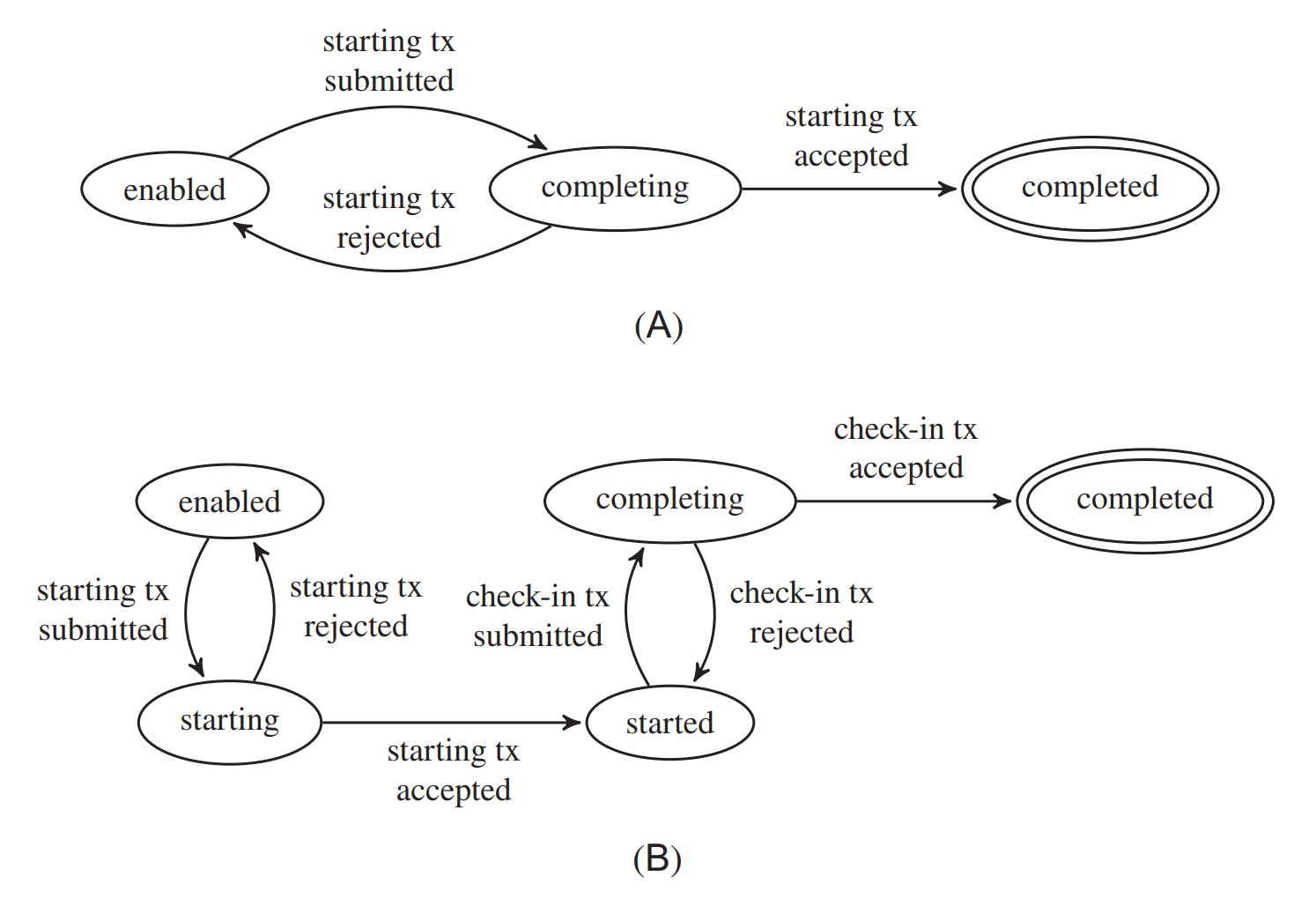


图 73 外部元素转换过程示意图

由BPMN流程而生成的Solidity合约是基于一个状态机来运行的，其状态转移函数是step函数，在后文中会详细说明。对于外部元素，MainConverter在主合约中生成一个complete为后缀的函数，和在worklist中生成一个以元素名为名称的函数、以start为后缀的函数、以requested为后缀的事件以模拟外部元素，其中每一个函数或事件的调用和触发都对应了外部元素生命周期中的一个阶段。本节以示例中的外部任务enter\_form\_link为例说明其生命周期中的各个状态。如5.4.3节的合约框架所示，在step函数中执行到该外部任务所对应的分支时，该任务进入enabled状态；随后step函数调用worklist合约中的enter\_form\_link\_start函数，参数为该任务执行的索引号，转入starting状态；worklist合约中的enter\_form\_link\_start函数执行会触发一个代表该外部任务的事件，即enter\_form\_link\_Requested事件，该事件的参数包括该任务执行的索引号，避免生成的Solidity执行逻辑与BPMN流程逻辑不一致。参数还包括BPMN流程图中指向该任务的数据对象export\_to\_task，其起到的作用是将BPMN流程中全局取的数据以事件的形式传递给外部用户。在该事件抛出之后，状态机转入started状态，等待外部用户向链上发送含有输入参数的交易以继续驱动状态机的执行；外部用户接收到来自链上运行抛出的事件后，解析该事件的参数，并由外部用户或程序根据该事件和相应的参数返回结果。该结果返回链上的方式是由外部用户调用worklist合约中的enter\_form\_link函数，并将外部返回的结果以参数的形式调用。该合约函数调用完成后，状态机转入到completeing状态；在worklist合约的enter\_form\_link函数中，先对调用的顺序和逻辑进行校验，校验后调用主流程合约中的enter\_form\_link\_complete函数以结束这一任务的执行，状态机转移至completed状态。

对于内部元素，由于其不涉及链下操作和与链下外部对象的交互，因此MainCoverter模块不会为外部元素生成额外的函数模拟其操作，所有操作都放在主流程合约中的状态转移函数中完成。以示例中的排他网关为例，其生成的代码片段如下：

1. function step(uint tmpMarking, uint tmpStartedActivities) internal {
2. while (true) {
3. if (tmpMarking & uint(16) != 0) {
4. emit execution\_progress("Gateway\_0h5r00z");
5. tmpMarking &= uint(~16);
6. if (\_criteria > 4)
7. tmpMarking |= uint(64);
8. else
9. tmpMarking |= uint(32);
10. continue;
11. }
13. ……
14. break;
15. }
16. if (marking != 0 || startedActivities != 0) {
17. marking = tmpMarking;
18. startedActivities = tmpStartedActivities;
19. }
20. }

图 74 排他网关转换生成的智能合约代码片段

内部元素的执行流程都放在了主流程合约中的状态转移函数step来完成，位于该内部元素所对应分支的代码段内。基于排他网关的性质，排他网关所有输出的条件顺序流都会被转换成if语句（如图 74中代码第6至9行）。因此，内部元素的执行均在主流程合约中完成。如图 75所示，结合内部元素的生命周期进行说明：当代表流程运行的令牌到达step函数中的if语句时，流程运行的状态机进入enabled状态；若此时的运行状态通过校验（图 74中代码第6行），则状态机转入completing状态，并对令牌的值赋值为下一个BPMN元素对应的语句位置；令牌赋值完成后，合约运行下一个BPMN元素转换生成的Solidity语句，状态机进入completed状态，对于下一个BPMN元素的运行来说，则下一个元素的状态机进入enabled状态。

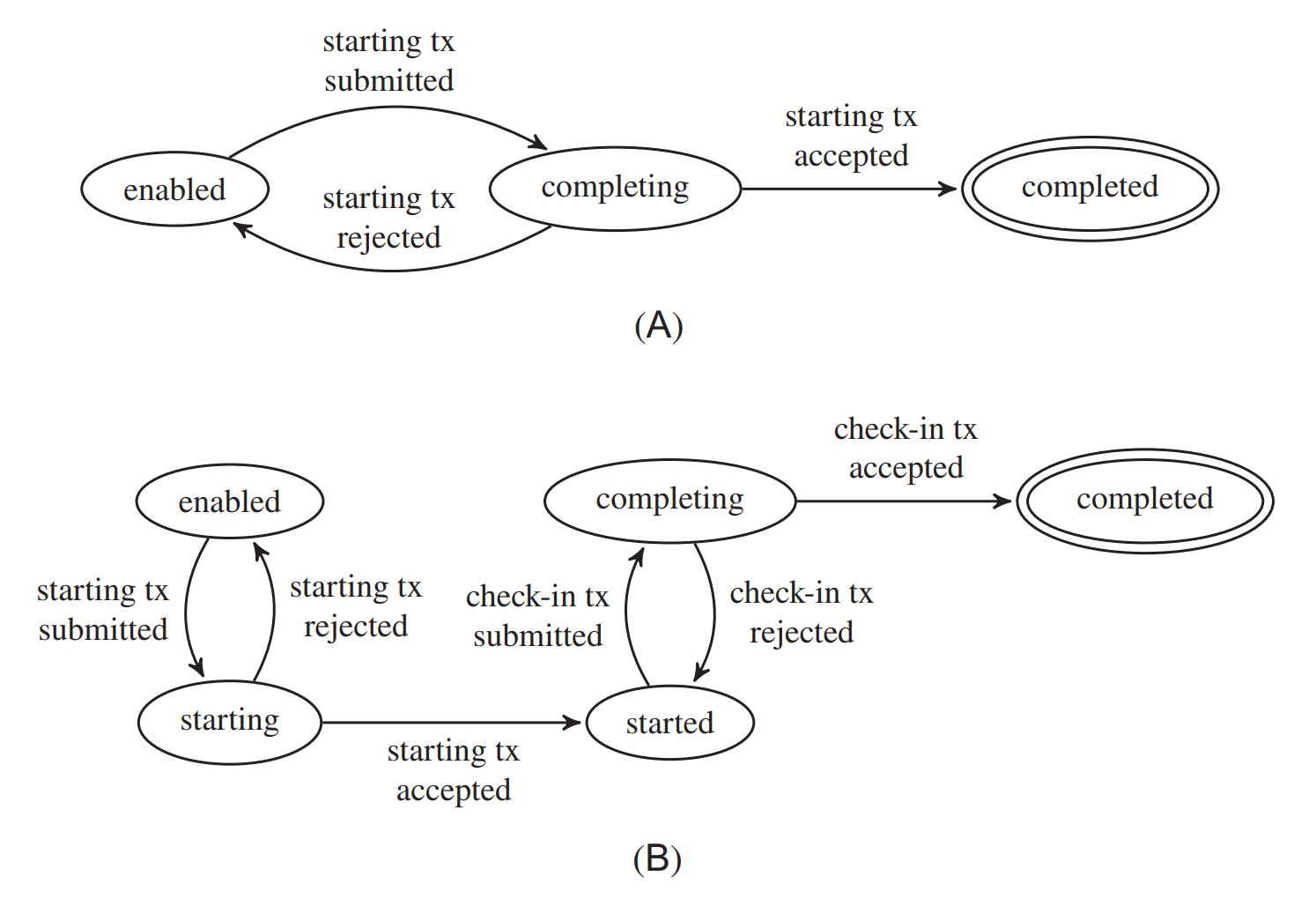


图 75 内部元素转换过程示意图

本段将对主流程合约中表示状态机当前状态的变量进行介绍，即生成合约框架中（图 72中代码）的第4、5行。在BPMN的规范中，控制流的模拟使用一个符合BPMN标准的令牌来进行。在一个流程实例创建后，开始事件会生成一个令牌，该令牌会遍历模型中的顺序流直至到达结束事件。为实现令牌模拟，需要为BPMN模型中的每个节点分配顺序索引(从1开始)。本文使用索引0作为流程的标识符。假设该流程在任何时候一个顺序流上最多只能有一个令牌，这样就可以使用一个位数组来编码流程实例在给定状态下的令牌分布。这样的位数组会被编码为256位无符号整数（EVM中的默认字长）。本项目生成的主流程合约包含两个整数变量marking和startedActivities，其用途是编码流程实例的当前状态。变量marking是一个位数组，用途是编码流程模型所含各顺序流之间的令牌分布。每个顺序流都有对应的一个位数：如有令牌则为1，否则为0。变量startedActivities编码的是触发的外部元素（如用户任务）的集合，其中的每一位都对应一个外部元素。另外，生成的worklist合约中动态数组processInstanceAddr存储的是每个可外部元素调用者的地址，在worklist合约中抛出代表该外部任务的事件后，worklist合约同样会记录抛出该合约的账户地址，并在回调该接口的时候对调用者地址进行验证，以确保该外部事件的状态已经到达了started状态，等待外部用户调用worklist合约中外部任务对应的接口使状态机从started状态转移至completing状态。在图 80所示代码第19行的约束条件，即通过processInstanceAddr存储的地址确保enter\_form\_link\_start函数已经被执行，避免状态机未经过started状态直接到达completing状态。

MainConverter模块使用基于逐位元的操作来处理所有对流程状态的查询和更新。例如使用逐位AND测试两个集合的包含关系，检查元素是否已启用或启动；用逐位OR将集合编码为整数，用途是在marking变量中追加令牌；还可组合使用NOT和AND从变量marking、startedActivities中移除令牌或元素。

图 76中代码展示的是控制流的一个实现样例，包括状态转移函数step，和主流程合约中用于切换状态（调用状态转移函数）的complete函数。下面以第1-10行的函数enter\_form\_link\_complete为例进行说明。当用户执行任务enter\_form\_link时，worklist合约会调用此函数。第5行要求tmpStartedActivities & uint(2) != 0就表示索引为2的活动（即enter\_form\_link\_start）已启动。这里的2是二进制数10的十进制表示（即位数组倒数第二位，索引位置1）。要注意的是，在任务启用时（即第14行条件tmpMarking & uint(2) != 0为真时），第15-18行的if分支内的语句必须与worklist合约中的start函数交互。因此，入向弧上的令牌会被删除，tmpMarking &= uint(~2)；该任务被启动，tmpStartedActivities |= uint(2)。这里的数字基本上是将模型编译为Solidity时分配给相应顺序流和任务的索引。

1. function enter\_form\_link\_complete(uint elementIndex, string memory link) external {
2. uint tmpMarking = marking;
3. uint tmpStartedActivities = startedActivities;
4. if (elementIndex == uint(1)) {
5. require(msg.sender == \_worklist && tmpStartedActivities & uint(2) != 0);
6. \_link = link;
7. step(tmpMarking | uint(4), tmpStartedActivities & uint(~2));
8. return;
9. }
10. }
11. function step(uint tmpMarking, uint tmpStartedActivities) internal {
12. while (true) {
13. if (tmpMarking & uint(2) != 0) {
14. emit execution\_progress("Activity\_1fwkk2f");
15. Process\_FormAudit\_Worklist(\_worklist).enter\_form\_link\_start(1);
16. tmpMarking &= uint(~2);
17. tmpStartedActivities |= uint(2);
18. continue;
19. }
20. if (tmpMarking & uint(4) != 0) {
21. emit execution\_progress("Activity\_1nik3oi");
22. Process\_FormAudit\_Worklist(\_worklist).get\_form\_start(2, \_link);
23. tmpMarking &= uint(~4);
24. tmpStartedActivities |= uint(4);
25. continue;
26. }
27. if (tmpMarking & uint(16) != 0) {
28. emit execution\_progress("Gateway\_0h5r00z");
29. tmpMarking &= uint(~16);
30. if (\_criteria > 4)
31. tmpMarking |= uint(64);
32. else
33. tmpMarking |= uint(32);
34. continue;
35. }
36. if (tmpMarking & uint(8) != 0) {
37. emit execution\_progress("Activity\_1a9g3hq");
38. bytes memory b\_\_form = bytes(\_form);
39. \_criteria = b\_\_form.length;
40. tmpMarking = tmpMarking & uint(~8) | uint(16);
41. continue;
42. }
43. if (tmpMarking & uint(64) != 0) {
44. emit execution\_progress("Event\_1dl1lri");
45. emit Event\_1dl1lri\_Message('form\_success');
46. tmpMarking &= uint(~64);
47. if (tmpMarking & uint(126) == 0 && tmpStartedActivities & uint(22) == 0) {
48. (tmpMarking, tmpStartedActivities) = handleEvent("form\_success", "Default", tmpMarking, tmpStartedActivities);}
49. continue;
50. }
51. if (tmpMarking & uint(32) != 0) {
52. emit execution\_progress("Event\_15z22w3");
53. emit Event\_15z22w3\_Message('form\_fail');
54. tmpMarking &= uint(~32);
55. if (tmpMarking & uint(126) == 0 && tmpStartedActivities & uint(22) == 0) {
56. (tmpMarking, tmpStartedActivities) = handleEvent("form\_fail", "Default", tmpMarking, tmpStartedActivities);}
57. continue;
58. }
59. break;
60. }
61. if (marking != 0 || startedActivities != 0) {
62. marking = tmpMarking;
63. startedActivities = tmpStartedActivities;
64. }
65. }

图 76 控制流的智能合约实现样例

图 76中代码第12-66行中名为step的函数负责的是新流程状态的计算。作为一个内部函数，外部参与者是不能调用step的。但是，只要外部实体的函数调用更新了主流程合约的状态，step都会被调用。本文再次使用图 76中代码第1-10行所示的函数enter\_form\_link\_complete来说明step函数的工作原理。在第7行中，函数step被调用来更新流程状态，同时还通过更新tmpMarking来使传出的顺序流生效，和通过更改tmpStartedActivites将enter\_form\_link标记为已完成。tmpMarking与tmpStartedActivities的值变更会以参数的形式被函数step接收。前述都完成后，step函数会根据当前标记识别出一组已启用的BPMN元素。事实上，函数step会反复执行一系列if语句来确定当前标记是否启用了某个BPMN元素。如发现有启用，则step函数就开始执行相应的元素。由于并发的存在，在给定状态下可能会启用不止一个元素。因此，函数step必须重新启动while循环直至无法发现任何其他已启用的元素。只有在这一时刻，新计算出的实例状态才会被存入合约变量marking和startedActivities。

### BPMN数据流转换过程

在BPMN中，数据流是用来表示信息或数据如何在业务流程中流动的元素之一，其包括了数据信息在不同活动、任务、决策点和网关之间的传递和交换方式。本节将对BPMN业务流程中数据流到Solidity智能合约的转换进行讨论。

图 78中的代码是本项目自动生成的智能合约的一部分，其对应的流程是图 79所示的用户提交表单流程。Process\_FormAudit\_Contract合约被称为主合约或流程合约，其中第12-14行定义的是流程中的变量，这些变量定义来自流程全局区定义的数据对象。第16-25行对应的是函数enter\_form\_link\_complete，其生成的来源是图 79示例中的enter\_form\_link用户任务。需要与外部资源（用户任务和服务任务）交互的任务可以读/写流程合约中的变量，对应数据映射在BPMN流程图中指定，规范为：

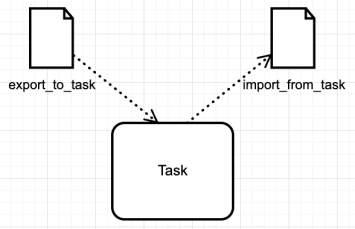


图 77 数据流转换过程示意图

在BPMN规范中，数据对象与任务的连接用虚线表示。export\_to\_task数据对象定义了任务从流程合约读取哪些变量（即任务的输入参数）。import\_from\_task指定的是任务的输出参数，即任务从外部资源获得的数据。随后本项目会将输出参数映射到全局变量。举例来说，enter\_form\_link任务的目标是让用户输入待提交表单的链接地址，因此不需要提供预先的信息给用户，所以没有import\_from\_task这一数据对象；enter\_form\_link任务返回的是用户提交的表单链接地址，该参数在合约代码中用参数link表示（与BPMN流程图中export\_to\_task数据对象的名称一致），通过类型检查后会被复制到全局区的\_link变量。

1. pragma solidity ^0.7.0;
2. contract Process\_FormAudit\_Contract {
3. uint public marking = 2;
4. uint public startedActivities = 0;
5. address internal \_worklist = address(0);
6. event Event\_1dl1lri\_Message(string messageText);
7. event Event\_15z22w3\_Message(string messageText);
8. event execution\_progress(string msg);
9. string \_link;
10. string \_form;
11. uint \_criteria;
12. function enter\_form\_link\_complete(uint elementIndex, string memory link) external {
13. uint tmpMarking = marking;
14. uint tmpStartedActivities = startedActivities;
15. if (elementIndex == uint(1)) {
16. require(msg.sender == \_worklist && tmpStartedActivities & uint(2) != 0);
17. \_link = link;
18. step(tmpMarking | uint(4), tmpStartedActivities & uint(~2));
19. return;
20. }
21. }
22. ……
23. }

图 78 用户提交表单流程生成的智能合约代码片段

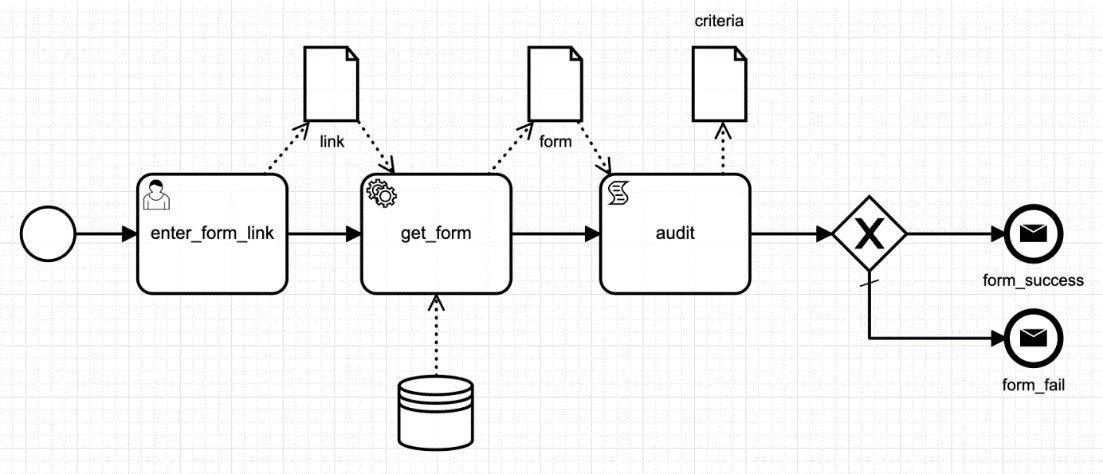


图 79 用户提交表单流程图

智能合约是无法直接调用外部资源的。为应对这一限制，MainCoverter转换而成的合约会在日志中发布Solidity事件，且这个日志会存入每个完整区块链节点并可以被外部程序读取。外部程序及其用户会对这些事件做出响应，并且以事务的形式将响应提交到区块链。采用这种交互方式的软件即被称为“预言机（Oracle）”。预言机基本上起到的是中介代理作用，在其两端的分别是主流程合约和外部应用程序，并且通常它还有另一个专门供其使用的合约（即Worklist合约）。图 80中的代码展示的是合约Process\_FormAudit\_Worklist的一部分，其作为中介来处理表单提交与审批流程中由用户任务生成的交互。用户可以触发的任何BPMN图中的任务（即用户任务）都会在合约中生成相应函数，接下来本文以enter\_form\_link任务为例来说明这些函数的生成方式以及外部数据在其中的传递流程。

1. contract Process\_FormAudit\_Worklist {
2. struct Workitem {
3. uint elementIndex;
4. address processInstanceAddr;
5. }
6. Workitem[] private workitems;
7. *// Events with the information to include in the Log when a workitem is registered*
8. event enter\_form\_link\_Requested(uint index);
9. event get\_form\_Requested(uint index, string outsideFlag, string outsideLink, string link);
10. function enter\_form\_link\_start(uint elementIndex) external {
11. workitems.push(Workitem(elementIndex, msg.sender));
12. emit enter\_form\_link\_Requested(workitems.length - 1);
13. }
14. function enter\_form\_link(uint workitemId, string memory link) external {
15. require(workitemId < workitems.length && workitems[workitemId].processInstanceAddr != address(0));
16. Process\_FormAudit\_Contract(workitems[workitemId].processInstanceAddr).enter\_form\_link\_complete(workitems[workitemId].elementIndex, link);
17. workitems[workitemId].processInstanceAddr = address(0);
18. }
19. ……
20. }

图 80 用于外部交互的Worklist合约代码片段

本项目在进行编译转换时，会为每个用户任务在工作列表（Worklist）合约中生成两个方法，即Process\_FormAudit\_Worklist中的enter\_form\_link\_start和enter\_form\_link。在主流程合约中，该任务会被实现为一个函数，即图 78所示代码中的enter\_form\_link\_complete。本项目生成前述这些函数的具体方式以及数据传递过程如下:

• <element name>\_start(<export\_to\_task>)。在主流程合约中的执行流到达相应元素时，由主流程合约调用此worklist合约上的函数。BPMN流程中形式为export\_to\_task的数据对象属于该方法的参数，其原本定义在BPMN流程图的数据对象里，并存储在主流程合约中的全局变量中。Worklist合约包含存储workitem的一个动态数组，如图 80所示其变量名是workitems。每个workitem 包含发起请求的流程合约的地址以及在流程合约中标识元素的索引，以保证编译生成的合约执行顺序与BPMN流程图中的逻辑一致。workitems数组可以被外部资源访问，在需要时也可在外部应用中检索，输入参数和存在动态数组中的workitem索引都存储在区块链日志的Solidity事件中，它们对外部应用是可见的。

• <element name>(<import\_from\_task>)。由外部资源调用该函数以提供继续执行该任务所需要的数据。提供的数据会以参数的形式被Worklist合约发送到主流程合约，完成后标记该workitem已完成（将地址设置为0）。需要注意的是，这里的外部资源必须符合该流程所使用的，已经定义好的任何访问控制策略，在生成的Worklist合约中使用require语句来进行约束。在图 80中代码显示的示例函数enter\_form\_link中，控制策略未对访问该函数的用户地址进行约束。

• <element name>\_complete(<import\_from\_task>)。这个函数由主流程合约实现，调用它的则是相关的Worklist合约。该函数会首先检查调用者的区块链地址是否匹配工作列表地址，后者就存储在流程合约的全局变量中（图 78所示代码第21行）。完成后任务被标记为“已完成”，同时流程实例的状态也被更新。之后工作列表调再调用该函数时会被拒绝。

服务任务在处理方式上类似于用户任务。二者的主要区别有一点，服务任务通常与网络、数据库服务相互交互；而在处理用户任务时，交互的数据流通常来自于用户输入。

# 参考文献

1. [] White, Stephen A. "Introduction to BPMN." *Ibm Cooperation* 2.0 (2004). [↑](#endnote-ref-0)
2. [] Paige, Richard F., Jonathan S. Ostroff, and Phillip J. Brooke. "Principles for modeling language design." *Information and Software Technology* 42.10 (2000): 665-675. [↑](#endnote-ref-1)
3. [] Schwitter, Rolf. "Controlled natural languages for knowledge representation." *Coling 2010: Posters*. 2010. [↑](#endnote-ref-2)
4. [] Ingolfo, Silvia, Alberto Siena, and John Mylopoulos. "Nòmos 3: Reasoning about regulatory compliance of requirements." *2014 IEEE 22nd International Requirements Engineering Conference (RE)*. IEEE, 2014. [↑](#endnote-ref-3)
5. [] Bajwa, Imran Sarwar, Mark G. Lee, and Behzad Bordbar. "SBVR business rules generation from natural language specification." *2011 AAAI Spring Symposium Series*. 2011. [↑](#endnote-ref-4)
6. [] Antoniou, Grigoris, and Antonis Bikakis. "Dr-prolog: A system for defeasible reasoning with rules and ontologies on the semantic web." *IEEE transactions on knowledge and data engineering* 19.2 (2006): 233-245. [↑](#endnote-ref-5)
7. [] Biryukov, Alex, Dmitry Khovratovich, and Sergei Tikhomirov. "Findel: Secure derivative contracts for ethereum." *Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers 21*. Springer International Publishing, 2017. [↑](#endnote-ref-6)
8. [] Lamela Seijas, Pablo, and Simon Thompson. "Marlowe: Financial contracts on blockchain." *International symposium on leveraging applications of formal methods*. Cham: Springer International Publishing, 2018. [↑](#endnote-ref-7)
9. [] Bartoletti, Massimo, and Roberto Zunino. "BitML: a calculus for Bitcoin smart contracts." *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018. [↑](#endnote-ref-8)
10. [] Ciatto, Giovanni, et al. "Blockchain-based coordination: Assessing the expressive power of smart contracts." *Information* 11.1 (2020): 52. [↑](#endnote-ref-9)
11. [] Grigg, Ian. "The ricardian contract." *Proceedings. First IEEE International Workshop on Electronic Contracting, 2004.*. IEEE, 2004. [↑](#endnote-ref-10)
12. [] Astigarraga, Tara, et al. "Empowering business-level blockchain users with a rules framework for smart contracts." *Service-Oriented Computing: 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings 16*. Springer International Publishing, 2018. [↑](#endnote-ref-11)
13. [] Parr, Terence J., and Russell W. Quong. "ANTLR: A predicated‐LL (k) parser generator." *Software: Practice and Experience* 25.7 (1995): 789-810. [↑](#endnote-ref-12)
14. [] Sandhu, Ravi, and Qamar Munawer. "How to do discretionary access control using roles." *Proceedings of the third ACM workshop on Role-based access control*. 1998. [↑](#endnote-ref-13)
15. [] Lindqvist, Hakan. "Mandatory access control." *Master's thesis in computing science, Umea University, Department of Computing Science, SE-901* 87 (2006). [↑](#endnote-ref-14)
16. [] Sandhu, Ravi S. "Role-based access control." *Advances in computers*. Vol. 46. Elsevier, 1998. 237-286. [↑](#endnote-ref-15)
17. [] Ouyang, Chun, et al. "From BPMN process models to BPEL web services." *2006 IEEE International Conference on Web Services (ICWS'06)*. IEEE, 2006. [↑](#endnote-ref-16)
18. [] Decker, Gero, et al. "BPEL4Chor: Extending BPEL for modeling choreographies." *Ieee international conference on web services (icws 2007)*. IEEE, 2007. [↑](#endnote-ref-17)
19. [] Tran, An Binh, Qinghua Lu, and Ingo Weber. "Lorikeet: A Model-Driven Engineering Tool for Blockchain-Based Business Process Execution and Asset Management." *BPM (Dissertation/Demos/Industry)*. 2018. [↑](#endnote-ref-18)
20. [] Steffen, Samuel, et al. "zkay: Specifying and enforcing data privacy in smart contracts." *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*. 2019. [↑](#endnote-ref-19)
21. [] Steffen, Samuel, et al. "Zeestar: Private smart contracts by homomorphic encryption and zero-knowledge proofs." *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022. [↑](#endnote-ref-20)