一种基于区块链智能合约的软件服务交易方法

王晟典[1)]，陈娥[1)]，朱岩[1)]，林映春[1)]，刘国伟[2)]

（1. 北京科技大学, 计算机与通信工程学院 北京 100083）

（2. 北京市经济和信息化局， 北京 100744）

摘　要：随着软件服务交易模式由提前付费向“先服务后结算”转变，软件即服务（SaaS）所依赖的订阅模式面临着软件服务金融化与法律化的挑战——既无法按实际使用量进行金融支付，也难以通过法律形式规范服务提供方、消费方、交易平台之间权利义务关系。据此，本文将智能法律合约（SLC）引入到服务计算平台中，提出一种服务即合约（SaaSC）架构。在法律化方面，SaaS+SaaSC 的组合支持 SLC 软件订阅合约中设立服务注册、发现、定制化三种条款，从交互动作、服务状态、状态转移流程等方面规范了各方当事人在服务注册、发现与消费三阶段的交互行为；在金融化方面，将服务接口声明添加到智能法律合约中，借助智能合约自动执行和检查条款实现了细化到服务接口调用级别的精准计费模式。进一步，以天气预报服务作为案例实现了基于区块链智能合约的在线软件服务获取、交付及合约化支付，验证了 SaaS+SaaSC 方案的合理性和有效性，表明软件服务合约化是一种新的可行技术路线。

关键词：区块链智能合约；SaaS；智能法律合约；微服务；服务注册；服务发现

中图分类号: TP319　　文献标识码：A

A Software Service Transaction Approach Based on Blockchain Smart Contracts

Wang Shengdian[1], Chen E[1], Zhu Yan[1], Lin Yingchun[1], Liu Guowei[2]

(1. School of Computer & Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China)

(2. Beijing Municipal Bureau of Economy and Information Technology, Beijing 100744, China)

Abstract: With software service transactions changing from pay-before-use to pay-as-you-go, the subscription model of Software as a Service (SaaS) is facing the challenges on legalization and financialization of software service. This means that it neither realize financial payment on a pay-as-you-go basis, nor regulate the rights and obligations among service’s providers, consumers and platforms in a legal form. To address these challenges, in this paper Smart Legal Contract (SLC) is integrated into service computing platform by introducing a new architecture, called Service as a Smart Contract (SaaSC). Firstly, a contract-type service interface scheme is designed to perform the subscription process of service registration and publication on SaaS. In this scheme, we define six types of interactions, four kinds of microservice’s states, and their state transition procedure, and then establish the mapping from general service interface following the OpenAPI Specification to the contract terms in the SLC-style SPESC language. Especially, a new term, called Service Registration Term (SRT), is proposed to attain a regularized interaction approach during service registration. In addition, the legal Negotiation-Acceptance mechanism is utilized for granting the consumer’s rights to obtain software service. Secondly, a payment mechanism for contracting consumers’ demand is proposed in the process of service discovery and consumption. Exactly, based on service matching approach with three-level cache, other new terms, called Service Discovery Term (SDT) and Service Customization Term (SCT), are designed to specify the requests and responses of service discovery and invocation. A billing model driven on SRT, SDT, and SCT is developed to implement fine-grained charging on the level of service interface calls and evidence preservation of service transactions in blockchain. Therefore, it provides a legal guarantee for the implementation of pay-as-you-go mode. To sum up, from the aspect of service legalization, the SaaS+SaaSC architecture supports establishing three kinds of terms, including service’s registration, discovery and customization terms, in SLC-based software subscription contract, so that a complete transaction procedure can be regulated among three above parties from their interactions, service states and their transition process. From the aspect of service financialization, the declaration of service’s

interface is appended into the SLC-based contract. By automatically executing smart contracts and checking the terms, the pay-as-you-go mode is implemented through fine-grained charging every time when calling service interface. Moreover, we take the weather forecast service as a case to implement and analyze the acquisition, delivery, and contractual payment of software service on blockchain smart contract. The experimental results demonstrate the feasibility and effectiveness of the proposed SaaS+SaaSC architecture, which should be a practicable approach for contracting of software services.

Key words: Blockchain Smart Contract; SaaS; Smart Legal Contract; Microservices; Service Registration; Service Discovery

1. 引言

软件即服务（SaaS）作为一种云计算服务架构，旨在帮助企业通过互联网交付应用程序，并交由第三方云供应商进行管理，已成为软件服务领域当前主流交易方法之一[1]。与软件外包开发、架构即服务（IaaS）、平台即服务（PaaS）一样，SaaS 本质上是一种将管理软件和实施服务一体化外包的服务模式，也是伴随着软件行业发展兴起的一种新型软件应用模式。它的商业成功很大程度上依赖于采用订阅模式制备软件许可证[2]。与传统的软件永久许可不同，订阅模式采用企业或消费者与 SaaS 提供商签署的一段时间（通常是每年或每月）的软件订阅合同（也称订阅协议），并在前者缴纳订阅费后向其授予 SaaS 相应的访问权限。此外，订阅模式采用不同缴费价格获得不同类型服务来满足不同用户的需求。这种方式增加了消费者服务选择灵活性、降低购置成本和试错成本、增强客户关系、以及 SaaS 提供商定价灵活性。

尽管软件订阅模式具有众多优势并代表着未来 SaaS 的必然选择，但是目前大多数订阅合同仍采用“提前付费”方式。这种方式会让 SaaS 提供商的现金流呈现出较好的状态，但服务费用与实际的服务数量与质量无关，因此无法按照实际使用量计费。这与按劳计酬的服务收费原则相违背，难以适应现代服务业自动化执行与监管的要求。更为合理的订阅模式是按照订阅合同和实际使用量让消费者“先服务后结算”或“先充值后扣费”，这种付费方式对于消费者来说成本控制和服务定制更加灵活。

然而，实现 SaaS“先服务后结算”方式远比提前付费方式复杂。这种复杂性体现在两个方面：首先，需要自动化的金融支付能力，保证依照订阅合同条款对租期、服务能力（如云虚拟机数目、计算性能、网络带宽）和服务质量进行自动化支付，避免人工干预；其次，需要强有力的法律化支持，保证服务过程中当事人履行订阅合同条款中义务的行为得到法律上的认可，服务过程的记录或存证具有法律效力，避免取证困难和因合同纠纷消耗律师大量人工成本[3]。

针对以上问题，本文提出一种服务即智能合约（Service as a Smart Contract, SaaSC）的新型软件服务架构，它是 SaaS 的一种商业化扩展，即以智能合约（Smart Contract）代码和平台为基础，通过自动执行的交易合约形式交付和支付软件服务，为软件订阅模式的实施提供更加有效的技术手段。其中，智能合约是个宽泛的计算机技术，它既包括部署在区块链（Blockchain）上、在满足预定条件时可自动执行并存证的计算机程序，也包括支持智能合约可执行程序开发、生成、部署、运行、验证的信息网络系统。交易合约则是法律上的概念，是指两方或多方当事人为完成某件事而共同遵循的约定，从而建立某种对当事人具有约束力的权利义务关系，如约定未来某个时间以一定数量金额支付某种软件服务的费用。这种交易合约形式为软件服务的“先服务后结算”方式提供金融化基础[4]和法律化保障[5]，进而智能合约系统为交易合约的自动执行提供技术保障。

智能法律合约（Smart Legal Contract, SLC）是一种具有法律约束力的智能合约，是含有合同构成要素、涵盖合同缔约方依据要约和承诺达成履行约定的计算机程序[6]。SLC 为法律合同文本和智能合约代码之间建立了转化的桥梁，保证了开放网络环境下合同的公平协商与交易合约的法律约束力，因此它已成为设计并开发具有法律效力智能合约的核心技术。为实现智能合约代码法律化，一种称为 SPESC 的面向法律语言已被提

出，它是一种类自然语言的语言规范[13]，旨在通过类自然语言、规范化的结构与书写、可自动化向可执行智能合约语言转换的方式来解决不同领域专家沟通、法律效力以及部分逻辑安全问题。进而本文将采用 SPESC 语言作为 SLC 的设计和开发工具。

为了开展 SaaSC 服务交易的实例化研究，本文将 SLC 与当前服务计算领域流行的微服务（Microservice）框架[7]相结合。微服务是一种开发软件的架构和组织方法，将独立服务通过明确定义的 API 进行组合，适用于云原生应用程序（如 SaaS）、无服务器计算或轻量级容器部署的应用程序[8]。微服务框架（如 Spring Cloud、Kubernetes）则为快速构建微服务提供注册、发布、发现及消费等一体化的支持，如 Spring Cloud 框架采用 Eureka 作为注册中心汇聚服务信息、统一管理和维护服务地址、快速发现与连接服务；采用 Spring Boot 打包和部署服务应用，支持 RESTful 接口传递服务数据。因此，微服务框架可为研究 SaaS+SaaSC 的“先服务后结算”软件订阅模式提供软件服务上的平台支撑。

针对目前 SaaS 架构难以依据现有法律合同规范当事人权利义务关系的问题，本文提出的 SaaSC 系统架构将面向法律合同的智能法律合约技术引入到服务注册、服务发现、服务消费三阶段中，为软件服务过程的法律化提供了一种有效解决方案，并为该 SaaSC 架构设计如下机制：首先，设计了一种服务注册与发布过程中的服务接口合约化方案，通过分析服务注册过程的 6 种交互动作、4 种微服务状态及 3 阶段状态转移流程，建立了遵循 OpenAPI 标准的服务通用接口到 SLC 语言中合约条款描述的映射关系，实现了一种基于智能法律合约“注册条款”的服务注册交互行为的规范化方法，以法律上“要约-承诺”方式为消费方合法获取软件服务权益奠定了基础。其次，提出了一种服务发现与消费过程中消费需求与计费合约化机制，以服务发现中的三级缓存机制和服务匹配方法为基础，设计了对服务发现请求和响应进行约定的智能法律合约“发现条款”与“定制化条款”，并在服务消费中借助智能合约自动执行并检查条款，实现细化到服务接口调用级别的精准计费模式，进而确保服务交易行为存证到区块链系统，从法律上为“先服务后结算”模式提供了保障。

2. 系统框架

本文的目标是探索“由软件到服务、再由服务到合约”的新型软件服务交易，即通过一种 SaaS+SaaSC 的软件架构实现“先服务后结算”的软件订阅模式，在技术上推动软件服务交易的法律化和金融化。架构核心是以服务质量和数量为基础，以按劳分配、公平公正的交易原则，建立“先服务后结算”的软件订阅模式，依法保护软件服务消费者合法权益。

2.1 系统架构

根据系统目标，本文设计了一种面向软件服务交易的 SaaS+SaaSC 架构。如图 1 所示，该架构在现有 SaaS

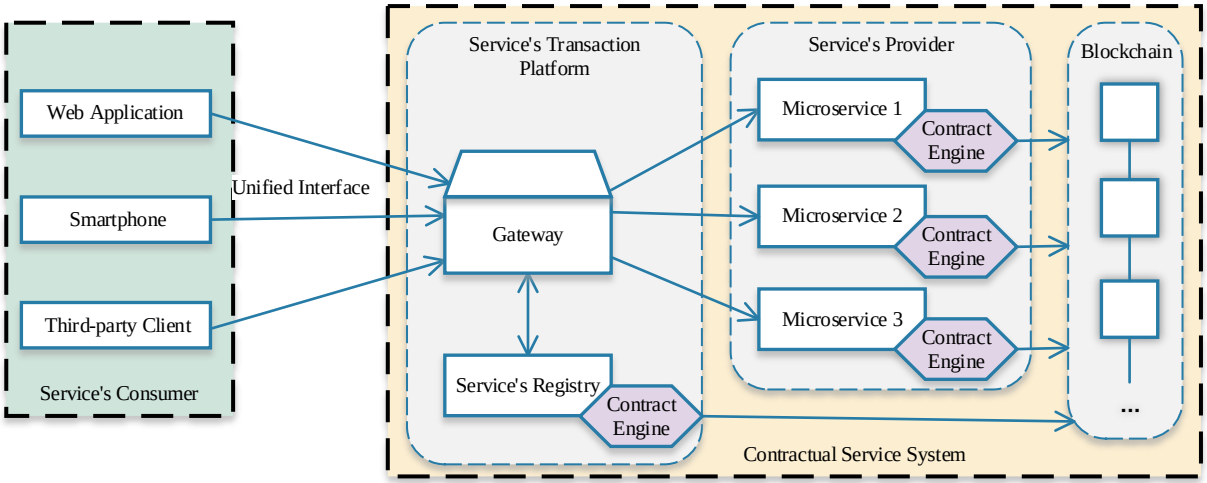

图 1 SaaS+SaaSC 架构示意图

Fig.1 Architecture of SaaS+SaaSC system

微服务框架基础上，借助区块链智能法律合约在法律化和自动化交易方面的独特优势，将软件订阅合同映射为智能法律合约，解决引言中基于智能合约的服务注册与发布及发现与消费中的关键挑战，将智能法律合约应用于服务计算环境。图 1 中 SaaS+SaaSC 架构的四个实体描述如下：

1） **服务消费方（Consumer）**：指为实现自身需求以有偿方式使用在线软件服务的一方，简称甲方；

2） **服务提供方（Provider）**：指为满足消费方需求，通过平台提供在线软件服务的一方，简称乙方；

3） **服务交易平台（Platform）**：支持服务提供方和消费方进行合约化服务交易的在线共享区域，简称丙方。

4） **区块链平台（Blockchain）**：指采用密码手段保障、只可追加、链式结构组织的分布式账本系统[10]，目的是实现去中心化服务交易的完整性、不可否认性、可追溯性[11]等安全目标。

在图 1 所示 SaaS+SaaSC 架构中，服务提供方与服务消费方通过合约平台签订由智能法律合约编写的“软件订阅合同”，进而产生服务消费活动。消费活动的发起方载体表现多样，可以通过 Web 应用、智能手机终端、第三方客户端等多种方式访问平台提供的统一网关接口（Gateway）。网关从注册中心（Service’s Registry）获取服务与地址对应信息，将服务请求路由转发到对应微服务接口。在此过程中，合约引擎（Contract Engine）作为区块链交互的执行模块，负责执行已签订“软件订阅合同”中相应条款，将合约代码和中间状态以区块链交易的格式（见表 1）读取或存入区块链。

2.2 系统实体关系与流程

在上述系统架构中本文将主要研究基于智能合约的服务注册、发现与消费三个阶段。为了清晰地展示这三个阶段，图 2 表示了主要实体间的交互关系，并将上述三大阶段按流程先后关系分为注册与发布（Registration and Publication）、发现（Discovery）、推荐（Recommendation）、请求（Request）、绑定（Binding）、消费（Consumption）、结算（Transation）等步骤：

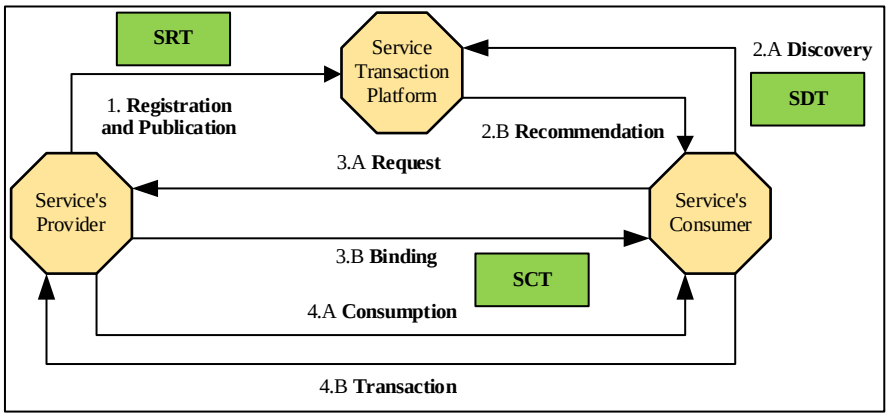

图 2 SaaSC 系统主要实体间关系

Fig.2 Relations among essential entities in SaaSC system

1） 服务注册与发布

本阶段是指服务将自身模块信息向交易平台宣称、平台再根据约定推广服务的过程。服务提供方使用 SLC 范本对服务注册与发布过程进行合同意义上的封装，被称为服务的合约化封装。在此过程中服务提供方提交接口注册信息的同时将服务接口承诺写入 SLC 范本，继而采用“服务注册条款 SRT”对服务接口宣称进行约定与检查（见图 2 阶段 1）。

2） 服务发现与绑定

消费方通过平台获取可用服务列表进而使用服务的过程称为服务发现与绑定。首先，平台作为联系服务提供方与消费方的桥梁，依据已注册 SLC 范本中“服务发现条款 SDT”接受发现请求，并提供对应的服务列表（见图 2 阶段 2.A）。其次，服务推荐是将多家服务整合，根据评价指标给出评估结果排名，由于服务推荐

使用现有推荐算法[12]，故在本文中不作为重点内容（见图 2 阶段 2.B）。然后，服务提供方和消费方根据定制化条款约定服务内容，消费方请求所选择服务授权软件许可（见图 2 阶段 3.A）。提供方根据定制化条款授权具体的服务资源绑定到条款指定的接口描述，双方通过合约订立确立最终合约形式并生成可执行智能合约程序部署到区块链平台（见图 2 阶段 3.B）。

3） **服务消费与结算**

本阶段是“先服务后结算”软件订阅模式的执行过程。根据条款规定的定价规则，服务提供方先期交付 SaaS 订阅使用权，消费方通过合约中规定的条款按需调用 SaaS 软件服务（图 2 阶段 4.A）。智能合约程序根据服务消费账单进行结算，将资金转账到服务提供方账户（图 2 阶段 4.B）。

2.3 软件订阅合同中的条款

上述阶段中，合约引擎提供智能合约部署与执行等功能。智能法律合约作为设计开发工具，约定平台与服务提供方、消费方之间采用三种条款规范两两之间交互行为与结果：

1） **服务注册条款 SRT**：是指合约中用于以资产（Asset）形式描述服务接口（Interface）并将资产提交给平台（通过 Commit 子条款）、以及平台将其中的服务接口发布到服务列表中（通过 Register 子条款）的相应条款，其中，Commit 和 Register 为合约模板中约定的两个子条款。

2） **服务发现条款 SDT**：是指合约中用于消费方向服务交易平台提交发现请求（通过 Request 子条款）及平台向消费方提供服务列表。依据条款中关于服务功能或质量的限制条件，服务发现请求中可附带检索信息以帮助平台缩小检索范围。

3） **服务定制化条款 SCT**：服务提供方和消费方依据协商一致的原则自行定义各方权利义务和服务计费标准。合同条款协商完毕的标志是合约订立。合约订立即“要约-承诺”的过程，与服务请求绑定过程相对应。

3. 服务注册与发布

本节将使用智能法律合约语言 SPESC 进行服务注册与发布的合约化封装。服务提供方提交接口注册信息同时将服务接口承诺写入智能法律合约，继而采用合约中资产表示对服务接口宣称进行约定与检查。SPESC 语法模型的构成要素包括合约名称、当事人描述、标的、合约条款、附加信息、合约订立等。同时，智能法律合约编写过程中涉及权利和义务、资产操作、表达式、时间表示等语法规范（见文献[13]）。后文将采用该语法模型对服务进行合约化描述。

3.1 合约当事人声明

首先，SPESC 智能法律合约中需要记录提供服务的负责人信息，服务实体作为合约当事人，对应智能法律合约的合约参与方（party）。在 SPESC 语言描述的智能法律合约中，合约当事人的属性采用键-值对描述，其中，属性键由合约范本指定，属性值可以留空或者预先填充。

本文中合约范本（或称合同范本）采用合约模板化思想[14]进行设计。该思想最早来源于李嘉图合约（Ricardian Contract）[15]，旨在通过可读性语言以类似合同文本的形式描述合同中每个条款的意思表示，并将其与当事人的意志选择分离开，前者被转化为合约模板，后者生成数据域。在合约模板代码被上传到区块链系统后，区块链提供的去中心化计算能力支持分布式环境下多方当事人对数据域进行填充与交换协商。因此，合约模板与数据域相分离的设计方式融合区块链去中心化计算能力，支持了多方当事人对合约进行并发操作。

如图 3(a)所示，合约范本为每个当事人设置账户和身份证明两个属性。由于服务提供方提供合约范本，故已在其中添加了属性值；其余当事人在合同协商过程中动态填入属性值。当事人还可以拥有其他属性，如可操作的动作等，在后文中将通过合约条款的方式进行声明。微服务应用（Service App）、当事人与智能合约的 UML 之间关联关系如图 3 (b)所示。其中，实体类（InstanceInfo）表示运行微服务的实例信息，微服务与

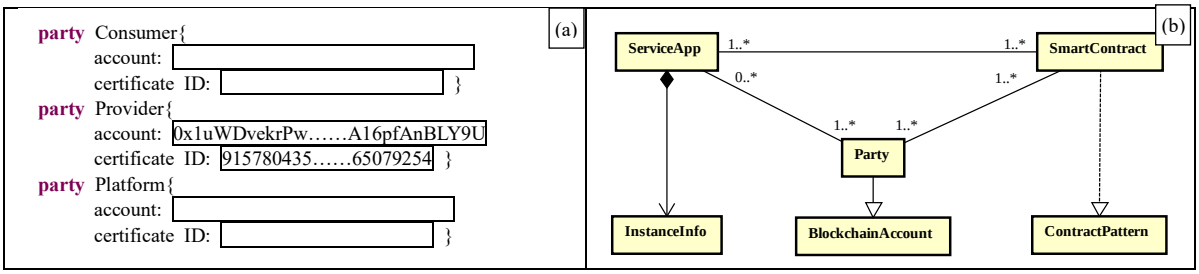

图 3 合约当事人描述与关系结构: (a)SPESC 合约当事人描述; (b)微服务、当事人与智能合约的 UML 模型

Fig.3 Contract party's description and relation model: (a) Example of parties' description in a SPESC contract; (b) UML relation model among microservices, parties and contracts

其实例信息为组合关系；Party 表示合约当事人，继承自区块链账户实体类（Blockchain Account）；智能合约的具体类（Smart Contract）继承自合约范本抽象类（Contract Pattern）。微服务与智能合约、当事人与智能合约均为多对多关系。当事人可作为提供方负责（零或）多个微服务，一个微服务至少需将一个当事人作为提供方，故两者间为一到多关系。此外，合约当事人在区块链上依然采用匿名机制（视为匿名矿工账户[15]），而由云平台负责对匿名区块链账户进行身份识别，确认其对应的真实身份，因此，智能合约发起区块链交易中记录该匿名帐户地址即可。

3.2 服务接口合约化

亨德里克森在《会计理论》中认为，用于购买服务的支出形成无形资产[17]。基于将 SPESC 合约中的软件服务承诺看作无形资产的前提假设，SPESC 语法定义的服务接口类型（type Service）继承于合约资产关键字（asset），见图 4 右部两个结构体的描述。这种定义方式的好处是将无形的服务与有形的资产进行统一封装，

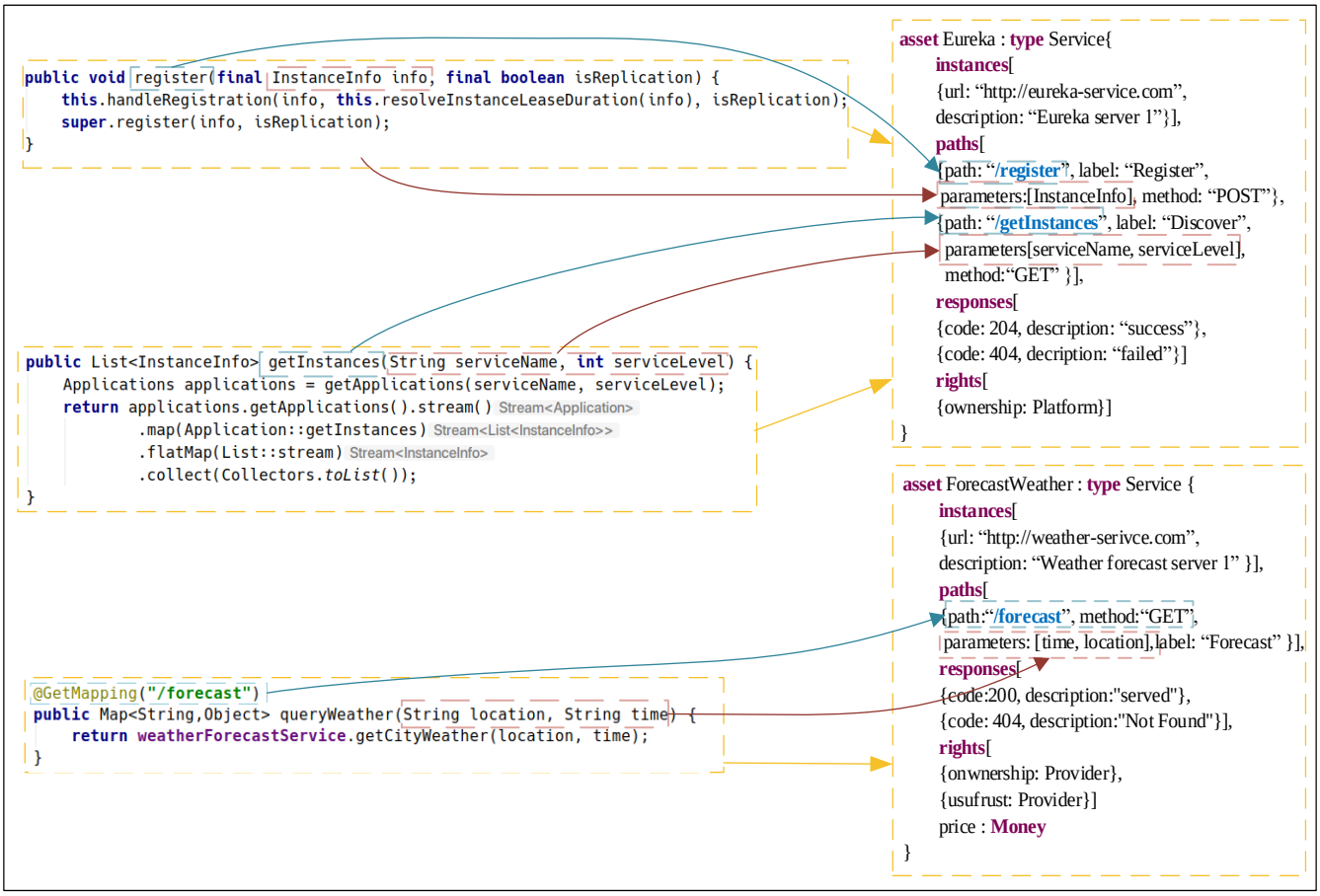

图 4 服务接口与智能法律合约中服务声明对应图

Fig.4 Mapping from service interfaces to service declarations in smart legal contract

并且符合软件工程的开闭模式设计原则。

图 4 中完整描述了样例中“服务注册接口”和“提供方服务接口”的合约化声明。服务接口（图 4 左部）采用 Java 语言编写，由 SPESC 定义的接口声明（图 4 右部）则由文档生成工具提取并转化为必要属性，转化关系如图中箭头所示。接口声明中的必要属性包括：提供服务实例的地址与服务名（instances）、服务所接收的路径（paths）及路径下接口所需参数（parameters）、动作类型（methods）及返回状态码（responses）等。服务接口的声明格式参照 OpenAPI 3.0 版本标准[18]实施。依据对服务接口的实际要求，合约当事人可协商添加原本服务接口描述中所没有的权属（rights）和必要的计费字段（price）等。基于上述合约化声明，提供方将添加接口描述后的合约范本与服务注册信息一并发送给注册中心，进而推广给服务消费方。

3.3 服务注册合约条款

在合同层面上，服务提供方（甲方）的注册过程应视为向服务交易平台（丙方）发起委托请求的过程，丙方的发布过程应视为接受甲方委托请求的过程。甲方与丙方之间这种服务注册发布过程的权利与义务关系使用如图 5 所的示条款进行规定。条款 no1 分为 2 个子条款，分别由甲方和丙方予以实施。子条款 no1_1 规定服务注册请求由甲方提交，其中所含的服务注册信息应包括：服务域名地址与端口、服务接口路径、生成时间戳、状态检查接口信息、以及被添加在自定义信息（metadata）中的合约范本。子条款 no1_2 规定丙方实施注册动作的义务，包括：

```
@@条款 1：甲方作为委托方，向丙方发送服务注册请求，丙方有义务为甲方发布提交注册的服务。
term no1_1：Provider can Commit(Service).
term no1_2：Platform must Register(Service)
        when Provider did Commit(Service)
        while grant Service::useRight to Platform
        where response::code is 204.
```

图 5 服务注册条款

Fig.5 Term of service registration

1） **when 语句**：指示前置条件，用于检查甲方服务信息是否已经提交。

2） **while 语句**：指示伴随动作，用于为丙方添加服务的使用权，以保证平台对服务功能进行测试和提供消费方试用的权利。

3） **where 语句**：指示后置条件，用于检查注册后的服务状态。

在 when 语句中，防止重复注册的原理是键-值表中的每个键只对应一条服务实例信息，不允许重复。服务提供方提交服务的操作过程中如果产生失败，合约引擎会回滚操作，以保证下次条款前置条件检查依旧会判定动作未执行。在 where 语句中，由丙方调用状态检查（healthCheck）函数访问甲方提供的地址，如果注册成功则返回成功状态码（即前述定义 code: 204）。

若服务通过条款 no1 的全部检查过程，则丙方将发布服务，完成服务注册与发布阶段。服务注册过程隐含的权利义务关系是服务提供方将服务授权给服务平台，平台接收委托则有义务为其推广销售。服务提供方与注册中心的交互与状态转移过程的技术点在下一节表述。

3.4 服务注册交互与状态转移

服务注册中心的任务是通过交互动作维护服务提供方微服务的注册信息，交互动作按发起方的不同可分为微服务主动发起和注册中心自行发起两大类。微服务能主动发起的动作有四种，分别是注册（Register）、心跳连接（Renew）、状态更新（StatusUpdate）、取消（Cancel），四种交互动作以 HTTP 报文格式封装。

注册中心能自行发起的动作有两种，分别是心跳续约超时（Expire）和定期清理（Evict）。在原有的 Eureka 模块中，注册中心还记录微服务的四种状态，包括服务启动（Starting）、服务上线（Up）、异常掉线（Out-Of-

Service）、服务下线（Down）。如图 6(a)所示，结合上述 6 种交互动作和 4 种微服务状态，对服务状态转移流程描述如下：

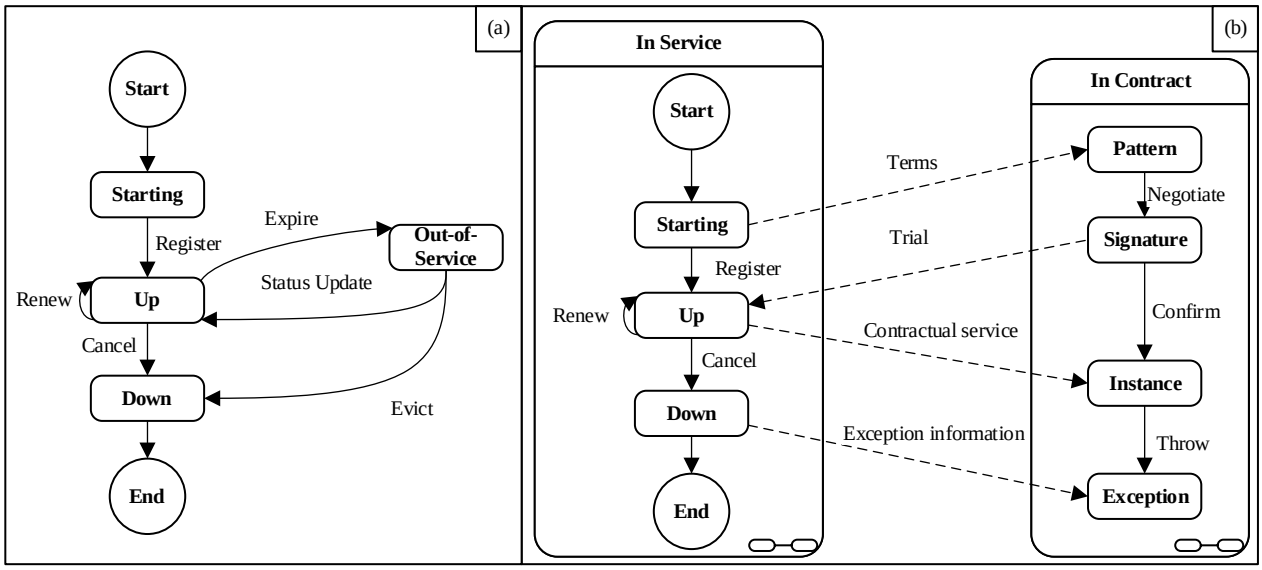

图 6 微服务注册状态转移示意图: (a)原有 Eureka 注册中心状态转移; (b)合约化注册状态转移

Fig.6 State transfer in microservice registration: (a) state transfer in original Eureka center; (b) state transfer in contract based registration

1）当注册中心收到微服务第一次注册请求时，将状态置为服务启动。如果通过审核成功，则将服务信息加入到服务列表中，状态更新为服务上线状态。

2）上线状态的微服务到达下一状态有 3 种可能，分别为维持上线状态、进入异常掉线状态和进入服务下线状态。为了维持上线状态，微服务需要定期向平台发送心跳连接以证实自身状态，未及时发送心跳连接的微服务将触发平台执行续约超时动作，被标记为异常掉线状态。微服务也可以主动结束生命周期，通过发送取消动作请求进入下线状态。

3）平台按清理周期（默认参数 eviction-interval-timer 设置为 60s）定期清理心跳超时的服务。清理周期设置一个较大的值以防止出现因注册中心网络中断而导致微服务误下线的情况，这属于注册中心的自我保护机制。被清理后的微服务进入下线状态，被动结束服务生命周期。

图 6(b)描述了合约化后的服务状态转移关系。结合服务部分与合约部分的状态转移关系，流程描述如下：

1）当收到微服务首次注册请求时，注册中心将该微服务状态置为服务启动；此后，状态服务商需要选择或提供含有服务定制条款的合约范本，完成服务注册条款约定的交互行为后才可上线。

2）消费方依据服务发现条款选择服务并进入合约签署过程，在双方确认完成合约订立后，合约部署到区块链系统。

3）依据双方协商的定制化服务条款与定价标准，消费者账户向合约账户充值生成合约实例后，该微服务进入合约化服务状态，实施使用时计费。

4）在合约化服务状态下，服务违约行为（心跳超时、条款执行失败、违约条款条件成立等）将触发合约引擎的异常处理功能。

相比图 6(a)原状态机模型，服务启动过程需要提供相应的服务注册合约条款信息，服务心跳超时会触发异常处理而不是直接进入服务异常状态。合约签署确认成功生成合约实例后，将进入合约化服务状态。上述合约实例是对 SPESC 合约条款逻辑的具体实现，合约条款执行过程包括检查前置条件、执行伴随动作和检查后置条件。根据合约交易的原子性（All or nothing）原则，只有当前置条件、动作执行和后置条件都满足后，合约状态才会产生更新。

4. 服务发现与绑定

SaaSC 架构下合约当事人依据智能法律合约的判定结果提供服务和支付服务费。服务提供方依据合约条款的规定提供服务，服务消费方则依据合约条款获取服务进而消费服务，并触发智能合约完成转账功能。消费方可根据实际需求检索服务承诺，通过平台签署智能法律合约以获取所需的软件服务。

4.1 服务发现合约条款

```
@@条款 2：乙方向丙方提交服务发现请求，丙方有义务向乙方提供服务列表。
term no2_1: Consumer can Request(Discover)
term no2_2: Platform must Discover
    when Consumer did Request(Discover)
    while grant Service::useRight to Consumer
    where response::ServiceName is WeatherForecast
        and response::ServiceLevel is good.
     where response::code is 204.
```

图 7 服务发现条款

Fig.7 Term of service discovery

如图 7 所示，服务发现条款规定服务消费方（乙方）可调用平台（丙方）的服务发现接口，丙方有义务返回匹配的服务列表。其技术原理是服务注册中心提供服务发现接口调用地址（图 7 左侧 getInstances 接口），消费方通过服务发现客户端发送 HTTP GET 请求，此过程触发合约引擎记录子条款 no2_1 中请求（Request）动作。服务发现（Discover）过程可以指定服务 ID 进行精准匹配，或者根据检索信息进行模糊匹配，包括：

1）**服务功能匹配**

服务功能匹配的种类包括：定购服务数量、时间长短、特定服务特征等，即请求中携带用户检索与上述种类相关的关键字，丙方根据关键字信息返回匹配后的相关列表。若服务提供方拥有多个地区的运营服务器，服务发现客户端可根据服务器所属地区提供对应服务实例的匹配结果。

2）**服务质量匹配**

匹配的服务将按照质量由高到低的顺序依次显示。服务质量评价指标根据平台统计数据收集，包括：该服务以往使用人数、是否产生违约、用户评分等。

合约引擎通过子条款 no2_2 监听注册中心的服务发现动作。该子条款由客户端 HTTP GET 请求触发，伴随动作将合约中记录的服务使用权授权给消费方。在动作执行结果产生返回值时检查是否返回成功状态码（即前述 code: 204），以保证客户端请求结果被成功响应。

服务发现阶段结束并进入下一阶段的标志是消费方为选定的服务预付款。预付款作为消费方履行服务约定的保证金交由平台暂存。完成预付款的过程需要服务参与方对合约条款声明达成一致，此阶段称为合约订立阶段。合约订立是服务参与方对服务合约中的意思表示进行“要约-承诺”的过程；在技术上，此阶段需采用密码学手段保障合约订立合规性，如使用身份认证技术验证参与方的身份使其不可伪造；采用数字签名保证当事人对签订服务合约行为不可抵赖；借助区块链存证以保证签署确认后的服务合约不可篡改。

4.2 服务发现缓存机制

服务发现是指服务消费方通过向平台发送检索请求获取所需服务的过程。平台负责维护已注册服务列表，该列表的数据结构为键-值对映射表。微服务架构的消费方无需事先知晓服务 IP 地址，而是交由服务发现客户端（图 8 中 Eureka Client）作为代理定期与注册中心同步缓存服务列表，通过检索服务缓存获取服务注册信息与实际 IP 地址等匹配信息。服务发现客户端缓存的服务列表需要与注册中心的服务列表保持最终一致，客户端缓存服务列表的获取包括以下两种模式：

1）**全量更新服务列表：**客户端请求全部应用的服务注册信息。如果注册中心的当前缓存中没有该信息，

则向上一级注册表请求得到服务注册信息，并且在当前一级添加缓存。当同步过程检查出列表中各个状态对应的服务数量不一致时，也会触发全量更新。

2）**增量更新服务列表：**在客户端已有缓存的前提下，通过定期比较得到本地缓存与注册中心缓存的差异获取某个应用的服务信息。缓存定期更新的过程由后台线程的驻留任务执行（默认周期值为⑥）。

如图 8 所示，注册中心提供三级缓存机制用于同步服务列表，分别为可读可写表（registry）、读写缓存表（readWriteCacheMap）、只读缓存表（readOnlyCacheMap），通过参数设置定期同步。可读可写表储存即时的全部服务列表数据；读写缓存表保存最近被读取或写入的服务列表（默认保存时间为①），服务发现客户端每读或写一条数据，可读可写表将这条数据即时写入读写缓存中；只读缓存表非即时更新，而是按配置周期从可读可写缓存表中更新（默认周期值为②）。针对服务发现场景对于及时性和吞吐量的不同需求，客户端可选择从读写缓存（默认配置③）或只读缓存（配置④）同步数据。

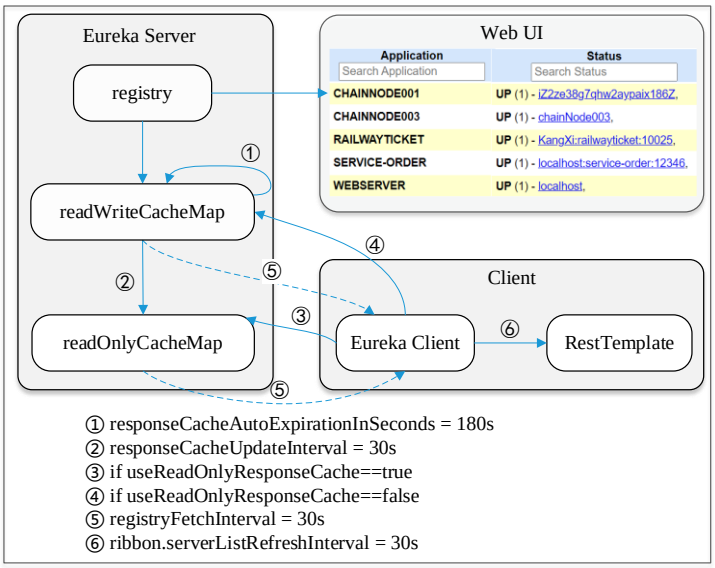

图 8 三级缓存同步机制示意图

Fig.8 Three levels of cache mechanism for replication

4.3 服务接口的请求绑定

请求绑定过程将协商完成的服务合约绑定到具体的传输协议上，与合约订立的“要约-承诺”过程相对应。由消费方发送服务授权请求，授权结果由服务提供方发布到区块链上进行存证。服务提供方和消费方通过区块链完成请求绑定后，在条款规定的服务要求内，服务提供方有义务提供按需服务，消费方有权利向服务提供方发起服务请求。

通常情况下，一次请求绑定过程由消费方发起，由提供方实现响应。目前的消息传输方式有 RESTful、RPC、消息中间件三种。RPC 传输数据格式通常为二进制格式，需要自定义协议格式；消息中间件虽然对异步传输支持较好，但需要单独部署服务端口，复杂度较高；而 RESTful 请求可采用封装了 JSON 格式的 HTTP 报文，可读性与可分析性更强。综合考虑，本文将采取 RESTful 接口传输 HTTP 数据报文。

如图 9 所示，智能法律合约中的服务接口声明在请求绑定的过程中采用 HTTP 报文封装形式。将接口声明中 url 字段与 path 字段映射为报文访问路径，将 method 字段映射为 HTTP 方法，将 parameters 字段映射为报文中的访问参数，将 responses 字段映射为接收 HTTP 报文的返回值等。最终实现将服务消费过程绑定到基于 HTTP 的请求响应行为中。

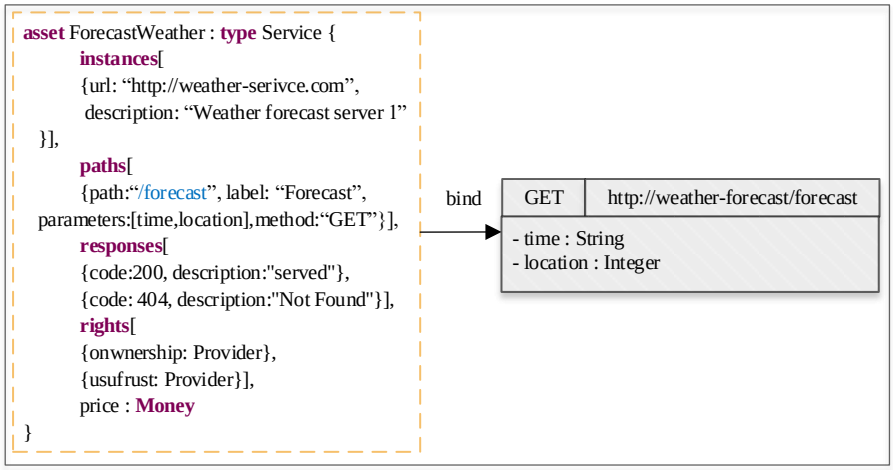

图 9 服务接口声明请求绑定至 HTTP 报文示意图
Fig.9 Mapping from service interface to HTTP package

5. 服务消费与结算

5.1 服务消费定制合约条款

在服务请求绑定过程中，消费方发起的合约签订请求可以看作服务要约[19]。请求授权的过程可以是多次协商获得的结果，提供方和消费方可以协商合约自定义条款中的具体服务参数，如服务期限、服务费用、付款时间、服务强度等。通信协商的过程可借助区块链实现存证。

双方对智能法律合约中的自定义条款协商达成一致，示例条款如图 10 所示，根据子条款 no3_1，消费方在发送服务消费的同时向合约存入押金作为预付款。根据子条款 no3_2，服务提供方在提供服务后从由合约转账取出预付款，实现“先服务后计费”。若服务调用结束返回成功状态码（即 code: 200）则调用付款接口，立即转账到服务提供方地址。智能合约将保证储存在合约中的余额非负。以上条款实现了以“先服务后计费”方式对服务接口进行按次、按量精准计费。

```
@@条款 3：甲方按次向乙方请求天气预报的结果并支付服务费。乙方成功返回服务结果后即取走费用。
term no3_1: Consumer can Request(Forecast)
    when serviceTimes < maxTimes
    while deposit price
    where balance >= price.
term no3_2: Provider must Forecast(time, location)
    when Consumer did Request(Forecast)
    while withdraw price
    where balance >= 0 and reponse::code is 200.
```

图 10 服务消费定制化条款
Fig.10 Term of service customized consumption

服务结束依赖于服务消费请求接收到 HTTP 状态码 code: 404，产生的原因分为两种情况：

1）**服务过程结束**：服务提供方依据消费定制条款中约定的条件提供一定次数或者一段时间的服务。每次或每段时间的服务履行记录由合约引擎以日志形式存证到区块链系统。当合约引擎执行条款条件不满足时（如超出服务次数或时间限制）将判断为服务过程结束，此时服务请求将不会响应，而是返回 404 状态码。

2）**合约终止**：合约有效期（如使用次数和期限等）结束将导致合约终止。此外，在服务期限内消费方退订服务也将导致合约终止，此过程需设置合约退订子条款，即当满足退订子条款的前置条件（如使用的最低次数或时间、合约账户中的剩余资金返还的比例计算方式等）时，平台将执行终止服务合约。终止的合约将无法进行实例化，网关从注册中心获取对应下线合约状态并拦截关联的服务消费请求，并返回 404 状态码。

5.2 智能合约的交易格式

软件服务的履行作为触发事件，发送给合约引擎执行智能合约中的服务条款，从而引发区块链中合约状态的改变。合约状态改变将以区块链交易方式予以记录、通信与存证，可用于合约交易结算的过程。

表 1 所示的区块链交易存储结构中合约相关的字段主要包括 Contract input 和 Contract output 两部分，其中，Contract input 部分记录一次操作收到的输入参数；Contract output 部分记录操作对结果的状态更新。将 Input 和 Output 存储在一个交易中以保证操作原子性。假设本次交易表示为T_i，前一次合约协商或执行的交易表示为T_{i-1}，则T_i的 lastTxid 字段为前一次合约动作产生的交易号，即与交易T_{i-1}的 txid 相等。此外，partySig 动作执行方签名与 partyPubList 中的公钥匹配，以对操作的当事人身份进行验证。

表 1 合约交易存储结构

Table.1 Structure of the transaction storage in contract

	Field	Description
Contract input	lastTxid	Transaction ID of the last one
	partySig	Signature of current action from the party
	actionType	Type of current action
	content	Name and parameters of current action
Contract output	txid	Current Transaction ID
	contractData	Storing data of latest contract state
	partyPubList	Public key list from the parties

6. 基于智能合约的服务案例设计与实验评估

本文平台系统选取现代服务业中应用广泛的天气预报服务场景作为应用进行案例与流程设计。之后介绍实验环境与方案，并给出系统运行过程和执行效率的实验结果。

```
contract ServiceCommission{
  party Consumer{
    account : 0xCA35b7d9Ee6068dFe2F4E8fa3c,
     certificate : ID. 989790571545851834l}
  party Provider{
    account : 0x1uWDvekrPwA16pfAnBLY9U,
     certificate  ID: 91578043565079254}
  party Platform{
    account : 0x4B0897b0f7E9C4e5364D2dB,
     certificate  ID: 0141609277513359}
  asset Eureka : type Service {
    instances[
    {url: "http://eureka−service.com",
    description: "Registration server 1"}],
    paths[
    {path: "/register", label: "Register" ,
    parameters:[InstanceInfo], method: "POST"},
    {path: "/getInstances", label: "Discover",
    parameters[serviceName, serviceLevel], method:"GET" }],
    responses[
    {code: 204, description: "success"},
    {code: 404, decription: "failed"}],
    rights[
    {ownership: Platform}]
  }
  asset ForecastWeather : type Service {
    instances [
    {url: "http://weather−serivce.com",
  description: "Weather forecast server 1" }],
    paths [
    {path:"/forecast", label: "Forecast",
       parameters:[time, location], method: "POST"}],
    responses [
    {code:200, description:"served"},
    {code:404, description:"Not Found"}],
    rights [
      {onwnership: Provider},
      {usufrust: Provider}],
    price : Money
    serviceTimes : Integer
  }
  balance : Money
  maxTimes : Integer

term no1_1: Provider can Commit(Service).
term no1_2: Platform must Register(Service)
  when Provider did commit(Service)
  while grant Service::useRight to Platform
  where response::code is 204.
term no2_1: Consumer can Request(Discover).
term no2_2: Platform must Discover
  when Consumer did request(Discover)
  while grant Service::useRight to Consumer
  where response::ServiceName is WeatherForecast
    and response::ServiceLevel is good.
term no3_1: Consumer can Request(Forecast)
  when serviceTimes <= maxTimes
  while deposit price
  where balance >= price.
term no3_2: Provider must Forecast(time, location)
  when Consumer did request(Forecast)
  while withdraw price
  where balance>=0 and reponse::code is 200.
Arbitration term: any controversy or claim arising out
  of or relating to this contract, or the breach
  thereof, shall be administered by institution:
  BeijingInternetCourt.
Contract conclusions:
  conclusion no1: This contract may not be modified in
    any manner unless in writing and signed by both
    parties.
  conclusion no2: Each of parties agrees with
    conversion from this contract to computer programs
    on smart contract platform, and approve that the
    programs' implementation has the same legal effect.
Signature of party Consumer {
  printedName: Service consumer,
  signature: 0x583031D1x1Ty13aD41,
  date: 2021/Apr/23rd}
Signature of party Provider {
  printedName: Service provider company,
  signature: 0x98e2a14F02302140225u6k,
  date: 2021/Apr/23rd}
Signature of party Platform {
  printedName: Service trading platform,
  signature: 0x87k2s8576BD6afaBfb7q5z,
  date: 2021/Apr/23rd}}
```

图 11 SPESC 智能法律合约天气预报服务案例

Fig.11 Complete Example of SPESC Smart Legal Contract for Weather Forecast Service

6.1 案例合约与流程设计

案例中使用 SPESC 语言描述的智能法律合约实现合同意思表示，如图 11 所示。服务提供方、消费方与

平台在服务注册、服务发现、服务消费三阶段的权利与义务，经三方共同参与协商、完善并达成一致。智能法律合约案例中将服务表达为一种资产类型，服务提供方拥有服务收益权，服务消费方拥有服务使用权。在合约中使用三个条款进行描述三方准许或必须的动作等。条款 1 至 3 分别规定了服务提供方与服务平台、服务消费方与服务平台、提供方与消费方之间的服务注册、发布、发现、消费交互过程。

以天气预报服务流程为例，本案例中的服务提供方和服务消费方在流程开始时已拥有平台的账户。服务提供方通过其账户发起服务注册，注册请求中附带合同范本。消费方在签署合同之前，享有服务的需求选择和试用权。消费方可在试用满意后选择服务，进入对应合约签署确认。当合同所有参与方都签署完毕后，最终合同书生成，发布到区块链上存证，保证可随时查取。

根据文献[20]中的方法，本案例采用代码翻译器将 SPESC 合同转化为可执行智能合约，由消费方预付定金以激活合约实例化运行。合约引擎通过执行合约接口调用函数记录提供服务和消费服务的过程。服务过程将由区块链系统记录并通过智能合约执行发起区块链交易实现自动计费。若合约账户中余额用尽，则平台将通过网关拦截服务接口调用，直到消费方用户继续充值续费后再恢复通过。若消费方用户不再希望续订天气预报服务，则平台将终止合约，由智能合约执行退订条款完成退款结算，结束合约化服务流程。

6.2 实验方案

依据前述的方法和案例，采用 ANTLR 和 Java ASM 框架实现 SPESC 语言在 JVM 运行的字节码编译器；基于 Spring Cloud Eureka 实现服务交易平台；使用 Spring Boot 微服务应用框架编写天气预报服务，提供 RESTful API 调用接口；基于多链区块链系统实现合约交易通信与存证。实验环境包含 3 台机器，其中 1 台为本地主机（Arch Linux, 8 核 Intel CPU@1.60GHz, 内存 8G），用于编译 SEPSC 智能法律合约和发起消费方调用；另 2 台为阿里云服务器（Ubuntu 16.04, 双核 Intel CPU@2.50GHz, 内存 1G），分别部署带有合约引擎依赖的 Eureka 平台与天气预报服务。

表 2 给出了一轮完整的基于智能法律合约的服务注册、发现、消费流程的模拟执行结果。表 2 中每一行记录了参与方的动作任务请求、触发的合约条款序列以及引发的响应动作。表 2 各列中，参与方动作请求列自上而下顺序记录了参与方的动作请求调用列表。智能合约参与方发起的动作请求按照合约规定触发条款条件检查后才能执行，通过执行合约伴随生成区块链转账交易进行链上资产转移操作。基于链上合约模板与数据域分离的设计原则，对于多用户和单用户操作的性能影响没有显著不同。

表 2 参与方动作任务请求列表

Table.2 Request list of parties' action task

Requested tasks	Invoked terms	Responsed actions
Provider commit(Register)	no1_1, no1_2	Platform Register(party, service)
Customer request(Discover)	no2_1, no2_2	Platform Discover()
Customer request(Forecast)	no3_1, no3_2	Provider Forecast(time, location)

6.3 实验验证

本轮实验分析验证了 SPESC 语言编译器执行效率与合约程序在微服务系统中的运行耗时。

表 3 智能法律合约编译实验

Table.3 Compiling experiment of the smart legal contract

Input		Compilation		Output
Number of bytes	Lines of code	Number of tokens	Average compiling time/ms	Number of bytes
994	118	187	570	2754

本地主机运行编译器对图 11 案例进行编译，输入合约文本通过 ANTRL 编写的规则解析生成语法树（AST），编译实验结果如表 3 所示，统计输入合约文本字符数为 994，行数为 118；编译过程中的 AST 含有标记数为 187 个；再通过 Java ASM 框架编写的字节码生成模块生成.class 可执行文件，重复执行 10 次的平均用时为 570ms，编译输出的可执行合约文件字节大小为 2754B。上述实验结果表明，合约编译速度快且所生成

的合约占用空间小。

可执行智能合约文件被打包为 jar 文件，通过微服务客户端发送事件请求模拟合约调用过程。按照表 2 顺序依次发送动作请求对接口测试，其中，每个接口分别进行 20 次调用测试并统计用时。结果如图 12(a)(b)(c)所示，图中横轴表示测试序号，纵轴表示测试用时。条形图中的加深部分表示加入条款检查之前的接口调用用时，三个接口调用平均用时分别为 3.80ms、11.40ms、9.60ms，数值记录在表 4 中第 3 列，并计算 95%置信区间填入第 4 列。图 12(a)(b)(c)斜线条纹部分分别表示添加执行条款 no1、no2、no3 后所增加的用时，斜线条纹与深色加和结果表示执行条款检查后的平均接口测试用时，分别为 5.35ms、13.00ms、11.20ms，将结果记录到表 4 第 5 列，同时计算 95%置信区间结果记录在第 6 列。

表 4 接口调用测试表

Table.4 Tests of interface time cost

Term No.	Service interface	Before executing terms		After executing terms	
		Average time/ms	Confidential interval/ms	Average time/ms	Confidential interval/ms
1	Commit/Register	3.80	[0.87, 6.73]	5.35	[0.91,9.79]
2	Request/Discover	11.40	[10.71,12.09]	13.00	[12.06,13.94]
3	Request/Forecast	9.60	[7.6, 11.6]	11.20	[9.12,13.28]

为了更清晰地评估上述实验结果，图 12(d)对数据进行统计分析，图中黑点表示接口平均用时，红色线段表示置信区间，经过比较执行条款检查前后的接口用时结果，计算出条款检查的运行时间对原有服务增加的比例分别为 40%、14%和 17%。相比于引入合约条款之前的情况，服务过程的时间消耗差距为常数级别，且随着原本服务耗时增加，合约条款检查对时间效率影响有减小的趋势。这说明在微服务系统中引入 SPESC 合约不会引起时间消耗产生数量级上的负担。

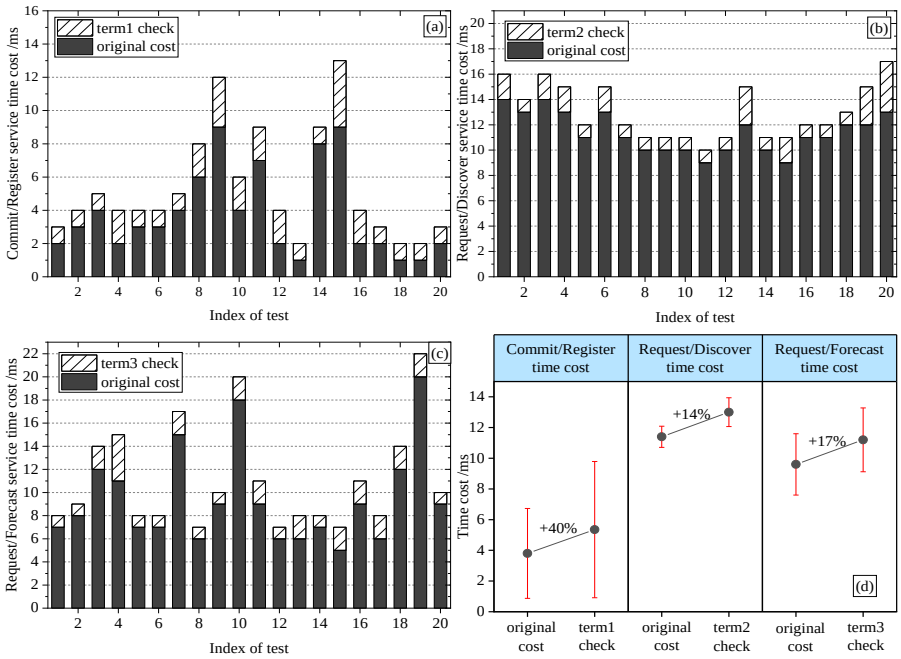

图 12 接口用时测试图: (a)条款 1 测试; (b)条款 2 测试; (c)条款 3 测试; (d)接口用时比较

Fig.15 Tests of interface time cost: (a)Test of term 1; (b)Test of term 2; (c)Test of term 3; (d)Comparison of interface time cost

7. 总结

本文提出了 SaaSC 架构，旨在优化 SaaS 预先付费的服务交易模型，形成 SaaS+SaaSC 的新型软件服务交易模式。该模式将服务订阅使用行为与智能法律合约中条款履行情况相绑定，可满足各方当事人生成服务合同用于履行服务的需求，更加方便自动化交易与监督监管。本文实验以案例方式对基于智能法律合约的软件服务交易模式进行验证，结果表明上述模式为软件服务的金融化和法律化提供一种新的可行技术路线。

基金项目

1. 国家科技部重点研发计划（编号 2018YFB1402702）

参考文献

[1] Varghese B, Buyya R. Next generation cloud computing: New trends and research directions. *Future Generation Computer Systems*, 2018(79): 849-861

[2] Bibi S, Katsaros D, Bozanis P. Business application acquisition: On-premise or SaaS-based solutions?. *IEEE Software*, 2012, 29(3): 86-93

[3] Chai Z G. Thoughts on Contract Law of Smart Contracts Under Blockchain. *Social Sciences in Guangdong*, 2019(04): 236-246
（柴振国. 区块链下智能合约的合同法思考. 广东社会科学, 2019(04): 236-246）

[4] Li R X. Research on the Application and Governance of Smart Contract in Financial Field. *Lanzhou Academic J*, 2020(06): 85-94
（李瑞雪. 智能合约在金融领域的应用及其治理研究. 兰州学刊, 2020(06): 85-94）

[5] Raskin M. The law and legality of smart contracts. *1 Georgetown Law Technology Review*, 2017, 1(2), 305-341

[6] Zhu Y, Wang D, Chen E, et al. Smart Legal Contract and its Research Progress. *Chin J Eng*, 44: 1-
（王迪,朱岩,陈娥,等. 智能法律合约及其研究进展. 工程科学学报, https://doi.org/10.13374/j.issn2095-9389.2021.02.22.001）

[7] Jamshidi P, Pahl C, Mendonça N C, et al. Microservices:The Journey So Far and Challenges Ahead. *IEEE Software*.2018,35(3):24-35

[8] Taibi, Davide, Josef Spillner, Konrad Wawruch. Serverless computing-where are we now, and where are we heading. *IEEE Software*, 2020, 38(1): 25-31

[9] Zou J, Wang Y, Lin K-J. A formal service contract model for accountable saas and cloud services // *Proceedings of 2010 IEEE International Conference on Services Computing*. Washington, 2010: 73-80

[10] Zhu Y, Wang Q S, Qin B H. Survey of blockchain technology and its advances. *Chin J Eng*, 2019, 41(11): 1361-1373
（朱岩, 王巧石, 秦博涵, 等. 区块链技术及其研究进展. 工程科学学报, 2019, 41(11): 1361-1373）

[11] QING S H. TTP Roles in Electronic Commerce Protocols. *J Software*, 2003, 14(11): 1936-1943
（卿斯汉. 电子商务协议中的可信第三方角色. 软件学报, 2003, 14(11): 1936-1943）

[12] Das D, Sahoo L, Datta S. A survey on recommendation system. *International Journal of Computer Applications*, 2017, 160(7): 6-10

[13] Chinese Institute of Electronics, People's Republic of China. T/CIE 159-2020 *Formal expression of blockchain smart contract*. Beijing: Standards Press of China, 2021
（中国电子学会. T/CIE 159-2020 区块链智能合约形式化表达. 北京：中国标准出版社, 2021）

[14] Clack C D. Smart Contract Templates: legal semantics and code validation. *Journal of Digital Banking*, 2018, 2(4): 338-352

[15] Grigg I. The Ricardian contract.//*Proceedings of 1st IEEE International Workshop on Electronic Contracting*. San Diego,2004:25-31

[16] Zhang K, Hu Y. On the Application of Smart Contract in the Framework of Contract Law.*Administration and Law*. 2020(03):108-116
（张可, 胡悦. 浅议《合同法》框架下的智能合约适用问题. 行政与法, 2020(03): 108-116）

[17] Hendriksen E S. *Accounting Theory*, R. D. Irwin, 1970

[18] Cao H Y, Falleri J R, and Blanc X. Automated generation of REST API specification from plain HTML documentation // *Proceedings of International Conference on Service-Oriented Computing*, Malaga, 2017: 453-461

[19] Chen J D. The Legal Structure of Smart Contract. *Oriental Law*, 2019(3): 18-29
（陈吉栋. 智能合约的法律构造. 东方法学, 2019(3): 18-29）

[20] Zhu Y, Qin B, Chen E, et al. An Advanced Smart Contract Conversion and Its Design and Implementation for Auction Contract. *Chin J Computer*, 2021, 44(3): 1-17
（朱岩, 秦博涵, 陈娥, 刘国伟. 一种高级智能合约转化方法及竞买合约设计与实现. 计算机学报, 2021, 44(3): 1-17）

[21] He X, Qin B, Zhu Y, et al. SPESC: A specification language for smart contracts // *Proceedings of 2018 IEEE 42nd Annual computer software and applications conference (COMPSAC)*. Tokyo, 2018, 1: 132-137.

[22] Zhu Y, Song W, Wang D, et al. TA-SPESC: Toward Asset-Driven Smart Contract Language Supporting Ownership Transa ction and Rule-Based Generation on Blockchain. *IEEE Transactions on Reliability*, 2021,70(3): 1255-1270.