Report 1A

System Specification

UCNS

University of Calgary Name Service

A Decentralized Agent-Based Naming Infrastructure

Course: SENG 696 - Agent-Based Software Engineering

Instructor: Professor Behrouz Far

Institution: University of Calgary

Authors:

Ali Mohammadi Ruzbahani [30261140], Shuvam Agarwala [30290444]

Fall 2025

Abstract

This report presents the comprehensive system specification for the University of Calgary Name Service (UCNS), a decentralized agent-based naming infrastructure deployed on the Polygon blockchain. UCNS demonstrates the application of agent oriented software engineering principles to build a production grade Web3 naming system. The architecture employs three autonomous smart contract agents: Registry Agent, Resolver Agent, and Pricing Agent, that coordinate through structured message passing to provide domain registration, ownership management, resolution services, and dynamic pricing. This specification details the system's purpose, scope, architectural design, functional requirements, and operational constraints, establishing a foundation for the subsequent GAIA based design and implementation phases. The system serves as both a practical decentralized application and an academic demonstration of multi agent system design in blockchain environments.

Contents

List of Figures

1 Introduction

1.1 Background and Motivation

The emergence of blockchain technology has fundamentally transformed how we conceptualize distributed systems, introducing trustless computation, immutable state, and decentralized coordination mechanisms. Traditional naming systems, such as the Domain Name System (DNS), rely on hierarchical centralized authorities that maintain registries, resolve queries, and enforce policies. While effective for decades, these systems introduce single points of failure, censorship vulnerabilities, and opacity in ownership and resolution logic.

The Ethereum Name Service (ENS) pioneered the concept of blockchain based naming by mapping human readable names to machine addresses using smart contracts. However, ENS and similar systems are often implemented as monolithic contracts with tightly coupled logic, making it difficult to isolate concerns, upgrade components independently, or reason about system behavior using agent oriented paradigms.

The University of Calgary Name Service (UCNS) addresses these limitations by explicitly adopting an agent-based software engineering approach. Rather than implementing the naming service as a single monolithic entity, UCNS decomposes the system into three specialized autonomous agents, each encapsulating distinct responsibilities and decision making processes. This architectural choice aligns with the principles taught in SENG 696, demonstrating how complex distributed systems can be designed, analyzed, and implemented using agent oriented methodologies.

1.2 Project Context

UCNS is developed as an academic project for SENG 696 - Agent-Based Software Engineering at the University of Calgary. The project aims to bridge theoretical agent oriented design principles with practical blockchain implementation, showcasing how autonomous agents can be realized as smart contracts that maintain state, enforce rules, and coordinate through message passing without centralized control.

The system is deployed on Polygon's Proof of Stake (PoS) mainnet, a Layer 2 scaling solution for Ethereum that provides fast transaction finality and low gas costs while maintaining full EVM compatibility. The choice of Polygon enables practical experimentation with production grade blockchain infrastructure while keeping operational costs manageable for an academic project.

1.3 Document Structure

This system specification is organized as follows: Section 2 establishes the system's purpose and high level objectives. Section 3 defines the scope and boundaries of the project. Section 4 presents the agent based architectural model, detailing each agent's responsibilities and interaction patterns. Section 5 enumerates functional requirements and system capabilities. Section 6 addresses non functional requirements including security, performance, and reliability. Section 7 describes the operating environment and deployment context. Section 8 outlines system assumptions. Section 9 acknowledges current limitations and future enhancement opportunities. Section 10 concludes with reflections on the agent-based approach and its implications for decentralized system design.

2 System Purpose

2.1 Primary Objectives

The University of Calgary Name Service exists to provide a fully autonomous, decentralized naming infrastructure that enables users to register, manage, and resolve human readable domain names on the blockchain. Unlike traditional naming systems governed by centralized authorities, UCNS operates entirely through autonomous smart contract agents that enforce deterministic rules, maintain tamper proof state, and coordinate without human intervention.

The system's primary objectives are:

1. **Decentralized Identity Management:** Enable users to establish verifiable on chain identities through self owned domain names ending with the ".ucns" suffix. These domains serve as portable identifiers that can be used across decentralized applications without relying on centralized identity providers.

2. **Autonomous Agent Coordination:** Demonstrate how specialized software agents with distinct responsibilities can collaborate to deliver complex functionality. The Registry Agent, Resolver Agent, and Pricing Agent operate independently & parallelly yet coordinate seamlessly through well defined interfaces.

3. **Transparent Pricing and Economics:** Implement dynamic pricing logic that calculates registration costs based on domain characteristics (length and duration) using deterministic on chain computation. All pricing parameters are publicly visible and verifiable.

4. **Verifiable Ownership and Resolution:** Provide cryptographically secured ownership records and resolution data that can be independently verified by any party. All ownership transfers and metadata updates are recorded immutably on the blockchain.

5. **Web2 & Web3 Integration:** Bridge traditional web interfaces with blockchain infrastructure, enabling users to interact with decentralized agents through familiar web based UIs while maintaining non custodial control through MetaMask wallet integration (A bridge web apps to blockchain).

6. **Academic Demonstration:** Serve as a comprehensive case study in agent-based software engineering, illustrating how theoretical concepts from SENG 696, including agent autonomy, communication protocols, coordination mechanisms, and distributed reasoning manifest in real world blockchain systems.

2.2 Target Users and Use Cases

UCNS serves multiple user categories with distinct interaction patterns:

Domain Registrants: Individual users or organizations seeking to establish blockchain based identities. These users interact with the system to register domains, configure resolution records, and manage ownership lifecycle.

Application Developers: Builders of decentralized applications (dApps) who need to integrate naming services for user friendly addressing. UCNS provides standard interfaces that dApps can query to resolve names to addresses.

Blockchain Researchers: Academics and researchers studying agent-based systems, blockchain infrastructure, or decentralized identity solutions. UCNS provides a transparent, well documented example of multi agent coordination in a production blockchain environment.

Web3 Enthusiasts: Users exploring blockchain technology who want hands on experience with decentralized naming without the complexity or cost barriers of larger naming systems.

Key use cases include:

- Registering a human readable name to receive cryptocurrency payments
- Associating profile metadata (email, social media, website) with a blockchain address
- Transferring domain ownership as part of asset trading or organizational handoffs
- Querying ownership and resolution data for identity verification
- Integrating naming resolution into smart contract logic or dApp interfaces

3 System Scope

3.1 Included Functionality

The scope of UCNS encompasses all essential operations required for a functional decentralized naming service. The system provides complete domain lifecycle management from initial registration through ownership transfer, including:

Domain Registration: Users can register available domain names by interacting with the Registry Agent and providing payment calculated by the Pricing Agent. Registration establishes ownership records with explicit expiration timestamps.

Ownership Management: The Registry Agent maintains authoritative ownership records, tracking the current owner, registration date, and expiration date for each domain. Ownership can be transferred explicitly by the current owner.

Resolution Services: The Resolver Agent stores and retrieves metadata associated with domains, including Ethereum addresses, email addresses, avatar URLs, descriptions, website URLs, and social media handles. Resolution queries return structured records that applications can consume.

Dynamic Pricing: The Pricing Agent computes registration costs based on domain length (shorter domains cost more) and registration duration (longer registrations cost proportionally more). Pricing tiers are configurable by the contract owner.

Availability Checking: Users can query domain availability before attempting registration. The Registry Agent evaluates whether a name is unregistered or has expired, making it available for registration.

Metadata Management: Domain owners can update resolution records at any time. The Resolver Agent verifies ownership through the Registry Agent before accepting updates, ensuring only authorized parties can modify metadata.

Web Interface: A comprehensive PHP based frontend provides user friendly access to all system functions through MetaMask wallet integration, enabling non technical users to interact with blockchain agents.

3.2 Excluded Functionality

To maintain focus on core agent-based design principles and academic objectives, UCNS intentionally excludes several advanced features common in production naming systems:

Decentralized Governance: The system does not implement DAO style governance mechanisms for collective decision making about pricing policies, reserved names, or protocol upgrades.

Automated Renewal: Domains must be manually renewed (through reregistration) before expiration. The system does not support automatic renewal through recurring payments or grace periods.

Domain Marketplace: While ownership transfer is supported, the system does not include built in marketplace functionality for listing, bidding on, or trading domains.

Subdomain Delegation: The current implementation supports only top level domains ending in ".ucns". Subdomain creation and delegation are not included.

Decentralized Storage Integration: Metadata is stored directly in contract state rather than using decentralized storage solutions like IPFS or Arweave.

Multi Chain Resolution: UCNS operates exclusively on Polygon. Cross chain name resolution or multi chain ownership records are not supported.

Advanced Records: The system supports a fixed set of resolution record types. Extensible record types or custom data structures are not currently implemented.

These exclusions allow the project to focus on demonstrating clear agent-based architecture while keeping implementation complexity manageable for academic purposes. Future phases could incorporate these features through modular extensions without disrupting the core agent design.

3.3 System Boundaries

UCNS operates within the following boundaries:

Blockchain Layer: All state changing operations occur on the Polygon PoS mainnet. The system does not maintain off chain databases or centralized servers for critical data.

Web Interface: The PHP based frontend serves as a convenience layer for user interaction but is not required for system operation. Advanced users can interact directly with smart contracts using web3 libraries or blockchain explorers.

Naming Convention: All registered domains end with the ".ucns" suffix. The system does not integrate with traditional DNS or other naming systems.

Character Set: Domain labels are restricted to lowercase alphanumeric characters and hyphens, with constraints on positioning and reserved names.

4 Agent-Based Architectural Model

4.1 Overview of Agent Paradigm

The UCNS architecture embodies the fundamental principles of agent oriented software engineering by decomposing the naming service into three autonomous entities, each exhibiting core agent characteristics: autonomy (independent operation without external control), social ability (communication and coordination with other agents), reactivity (perception and response to environmental changes), and pro activeness (goal directed behavior).

In the context of blockchain smart contracts, agents manifest as self contained contracts with:

- **Internal State:** Private storage variables maintaining agent knowledge.
- **Decision Logic:** Functions encoding autonomous reasoning and rule enforcement.
- **Communication Interfaces:** Public functions and events enabling inter agent message passing.
- **Authorization Mechanisms:** Access control ensuring appropriate agent boundaries.

This architectural approach contrasts with monolithic smart contract design by explicitly separating concerns, enabling independent agent evolution, and creating clear accountability for each system function.

4.2 Registry Agent

4.2.1 Agent Identity and Purpose

The Registry Agent (implemented as `UCNSRegistry.sol`) serves as the authoritative source for domain ownership and lifecycle state. This agent acts as the system's trusted ledger, maintaining the ground truth about which addresses own which domains and when those domains expire.

4.2.2 Agent Responsibilities

The Registry Agent autonomously manages:

1. **Ownership Records:** Maintains mappings from domain hashes to ownership structures containing owner address, registration timestamp, expiration timestamp, resolver address, and expiration flag.
2. **Availability Determination:** Evaluates whether a domain is available for registration by checking existence and expiration status.
3. **Registration Processing:** Accepts registration requests, validates naming rules, coordinates with Pricing Agent for cost calculation, verifies payment, and atomically records ownership.
4. **Ownership Transfers:** Processes explicit ownership transfer requests from current owners to new addresses.
5. **Authorization Verification:** Provides interfaces for other agents (particularly the Resolver Agent) to verify ownership and authorization for operations.
6. **Naming Rule Enforcement:** Validates domain names against character set restrictions, length limits, and reserved name lists.

4.2.3 Agent State and Knowledge

The Registry Agent maintains several internal knowledge structures:

```
struct DomainRecord {
    address owner;
    uint256 registrationTime;
    uint256 expirationTime;
    address resolver;
    bool isExpired;
}

mapping(bytes32 => DomainRecord) domains;
mapping(bytes32 => address) domainApprovals;
mapping(address => mapping(address => bool)) operatorApprovals;
```

Listing 1: Registry Agent State Model

This state enables the agent to reason about domain ownership, authorization delegation, and temporal validity independently.

4.2.4 Agent Decision Making

The Registry Agent's decision logic follows deterministic rules:

Algorithm 1 Registry Agent: Domain Registration Decision

```
Require: $domainName$, $duration$, $payment$
Ensure: Registration succeeds or reverts
 1: $hash \leftarrow keccak256(domainName)$
 2: if $isReservedName(domainName)$ then
 3:     revert "Reserved name"
 4: end if
 5: if $\neg isValidName(domainName)$ then
 6:     revert "Invalid name format"
 7: end if
 8: if $domains[hash].owner \neq 0$ and $\neg isExpired(hash)$ then
 9:     revert "Domain unavailable"
10: end if
11: $requiredCost \leftarrow PricingAgent.calculateCost(domainName, duration)$
12: if $payment < requiredCost$ then
13:     revert "Insufficient payment"
14: end if
15: $domains[hash].owner \leftarrow msg.sender$
16: $domains[hash].registrationTime \leftarrow block.timestamp$
17: $domains[hash].expirationTime \leftarrow block.timestamp + duration$
18: emit DomainRegistered($domainName, msg.sender, expirationTime$)
```

4.3 Resolver Agent

4.3.1 Agent Identity and Purpose

The Resolver Agent (implemented as `UCNSResolver.sol`) specializes in mapping domain names to resolution data. This agent maintains the connection between symbolic names and concrete metadata, enabling applications to discover addresses and associated information for registered domains.

4.3.2 Agent Responsibilities

The Resolver Agent autonomously manages:

1. **Address Resolution:** Maps domain hashes to Ethereum addresses for payment receiving and identity purposes.
2. **Metadata Storage:** Maintains extended profile information including email, avatar, description, website, Twitter handle, and GitHub username.
3. **Record Updates:** Processes update requests from domain owners, coordinating with Registry Agent for authorization verification.
4. **Multi Type Resolution:** Provides specialized query interfaces for different record types (address, text records, content hashes).
5. **Authorization Enforcement:** Rejects unauthorized update attempts by consulting Registry Agent ownership records.

4.3.3 Agent State and Knowledge

The Resolver Agent maintains resolution mappings:

```
struct ResolverRecord {
    address ethAddress;
    string email;
    string avatar;
    string description;
    string website;
    string twitter;
    string github;
}

mapping(bytes32 => ResolverRecord) records;
address public registryContract;
```

Listing 2: Resolver Agent State Model

4.3.4 Agent Decision Making

The Resolver Agent coordinates with the Registry Agent before accepting updates:

Algorithm 2 Resolver Agent: Record Update Decision

```
Require: $domainName$, $recordType$, $newValue$
Ensure: Update succeeds or reverts
 1: $hash \leftarrow keccak256(domainName)$
 2: $owner \leftarrow RegistryAgent.getOwner(hash)$
 3: if $owner \neq msg.sender$ then
 4:     revert ”Not authorized”
 5: end if
 6: if $RegistryAgent.isExpired(hash)$ then
 7:     revert ”Domain expired”
 8: end if
 9: $records[hash][recordType] \leftarrow newValue$
10: emit RecordUpdated($domainName$, $recordType$, $newValue$)
```

4.4 Pricing Agent

4.4.1 Agent Identity and Purpose

The Pricing Agent (implemented as `PricingAgent.sol`) encapsulates economic logic independently from ownership and resolution concerns. This agent calculates registration costs based on domain characteristics, enabling dynamic pricing without modifying registry or resolver contracts.

4.4.2 Agent Responsibilities

The Pricing Agent autonomously manages:

1. **Cost Calculation:** Computes registration costs based on domain length and duration using configurable price tiers.
2. **Pricing Policy:** Maintains tiered pricing structures where shorter domains cost more due to higher perceived value.
3. **Parameter Management:** Allows owner controlled updates to pricing tiers without affecting other system components.
4. **Transparent Computation:** Provides view functions enabling users to preview costs before committing to registration.

4.4.3 Agent State and Knowledge

The Pricing Agent maintains pricing configuration:

```
mapping(uint256 => uint256) public basePrices; // length tier ->
    base price
uint256 constant PRICE_PER_YEAR = 0.01 ether;
// Example tiers:
// length 1: 1.0 MATIC
// length 2-3: 0.5 MATIC
// length 4-5: 0.1 MATIC
// length 6+: 0.05 MATIC
```

Listing 3: Pricing Agent State Model

4.4.4 Agent Decision Making

The Pricing Agent's calculation logic is deterministic:

Algorithm 3 Pricing Agent: Cost Calculation

```
Require: $domainName, durationYears$
Ensure: $totalCost$
1: $length \leftarrow strlen(domainName)$
2: $basePrice \leftarrow getBasePriceForLength(length)$
3: $durationCost \leftarrow durationYears \times PRICE\_PER\_YEAR$
4: $totalCost \leftarrow basePrice + durationCost$
5: return $totalCost$
```

4.5 Inter Agent Communication

4.5.1 Communication Protocol

UCNS agents communicate through structured function calls and events, following a message passing paradigm that aligns with agent oriented communication protocols. While not implementing FIPA ACL explicitly (as Solidity lacks direct support), the interaction patterns follow similar principles.

4.5.2 Message Types

Table 1: Inter Agent Message Types

Message Type	Sender	Receiver
Cost Query	Registry Agent	Pricing Agent
Ownership Query	Resolver Agent	Registry Agent
Expiration Query	Resolver Agent	Registry Agent
Registration Event	Registry Agent	External Observers
Update Event	Resolver Agent	External Observers

4.5.3 Communication Flows

Figure 1 illustrates the complete registration communication sequence:

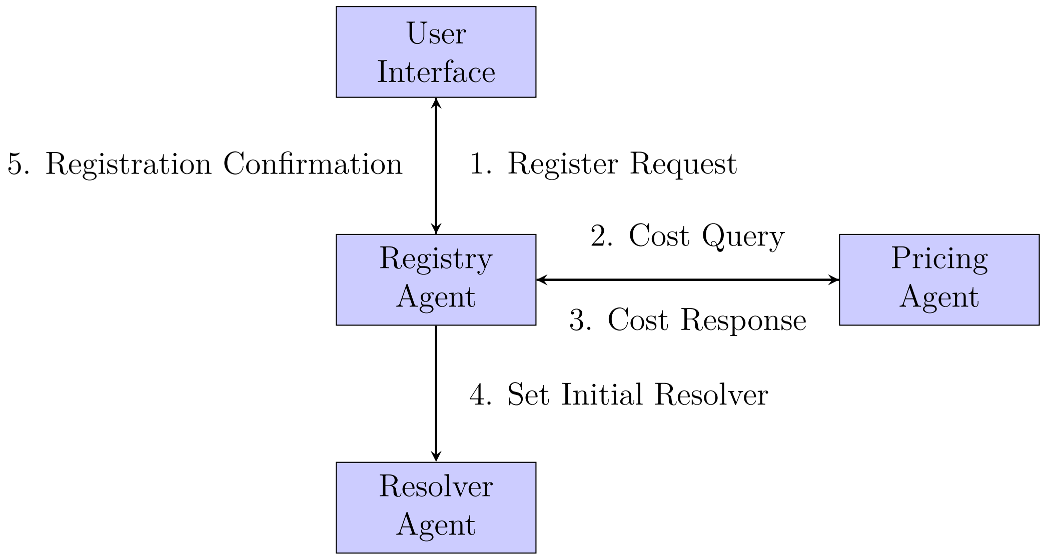

Figure 1: Domain Registration Communication Flow

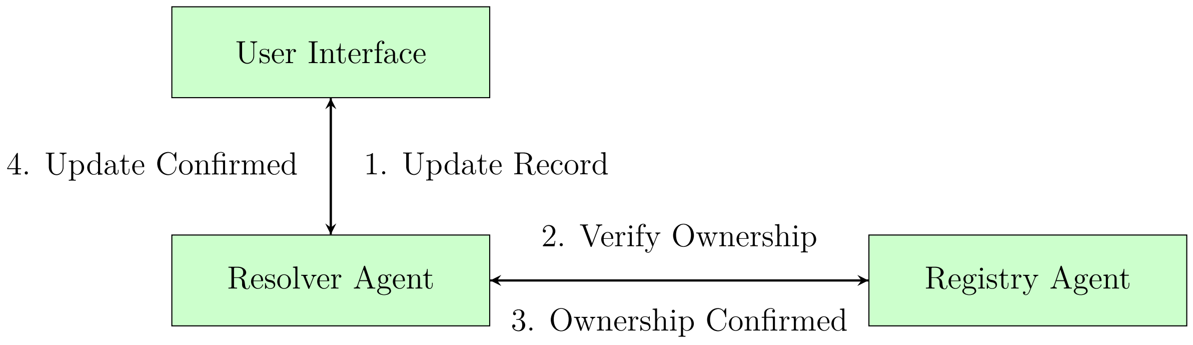

Figure 2: Record Update Communication Flow

4.6 Agent Autonomy and Independence

Each agent exhibits autonomy through:

Independent State Management: Agents maintain their own storage without direct access to other agents' internal state.

Self-Contained Logic: Decision-making occurs within agent boundaries based on local knowledge and explicit inter-agent queries.

Voluntary Coordination: Agents coordinate through clearly defined interfaces rather than shared state manipulation.

Upgrade Independence: In principle, individual agents can be upgraded or replaced without affecting others (within the constraints of immutable blockchain contracts).

This architectural separation enables reasoning about agent behavior independently and facilitates formal verification of agent properties.

5 System Functionalities

5.1 Domain Availability and Validation

5.1.1 Availability Checking

The system provides real time availability checking through the Registry Agent. Users can query whether a desired domain name is available for registration before initiating a transaction. The availability determination considers:

- Whether the domain hash exists in the registry
- If the domain has expired (expiration timestamp < current block timestamp)
- Whether the name violates naming rules or appears on the reserved list

5.1.2 Naming Rules

UCNS enforces strict naming conventions to ensure consistency and prevent abuse:

Table 2: Domain Naming Rules

Rule	**Specification**
Character Set	Lowercase a-z, 0-9, hyphen (-)
Length	Minimum 1, Maximum 64 characters
Leading/Trailing	No leading or trailing hyphens
Reserved Names	admin, ucns, root, system, null, void
Case Sensitivity	Case insensitive (normalized to lowercase)

5.1.3 Name Validation Algorithm

Algorithm 4 Domain Name Validation

```
Require: $domainName$
Ensure: Valid or Invalid
1: if $length(domainName) < 1$ or $length(domainName) > 64$ then
2:     return Invalid
3: end if
4: if $domainName[0] =' -'$ or $domainName[length - 1] =' -'$ then
5:     return Invalid
6: end if
7: for $char$ in $domainName$ do
8:     if $\neg(isLowercase(char)$ or $isDigit(char)$ or $char =' -')$ then
9:         return Invalid
10:    end if
11: end for
12: if $domainName \in$ {admin, ucns, root, system, null, void} then
13:     return Invalid
14: end if
15: return Valid
```

5.2 Domain Registration

5.2.1 Registration Process

Domain registration follows a multi step process coordinating Registry and Pricing agents:

1. **Validation:** User submits domain name and desired registration duration
2. **Availability Check:** Registry Agent verifies name is available
3. **Cost Calculation:** Pricing Agent computes required payment
4. **Payment Verification:** Registry Agent confirms sufficient MATIC provided
5. **Ownership Recording:** Atomic state update records ownership and expiration
6. **Event Emission:** Registration event broadcast for external monitoring

5.2.2 Registration Parameters

Table 3: Domain Registration Parameters

Parameter	Type	Constraints
Domain Name	string	Valid according to naming rules
Duration	uint256	1-10 years (in seconds)
Payment	MATIC	Must equal or exceed calculated cost
Resolver	address	Optional, defaults to system resolver

5.2.3 Registration State Transition

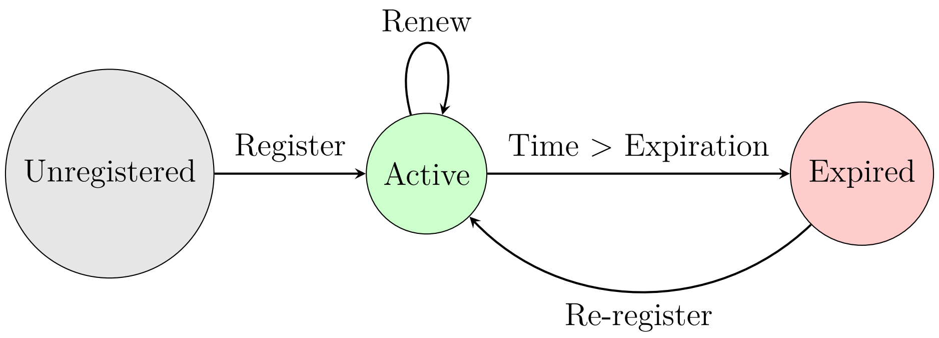

Figure 3: Domain Lifecycle State Machine

5.3 Ownership Management

5.3.1 Ownership Transfer

Registered domain owners can transfer ownership to another Ethereum address. The transfer process:

1. Current owner initiates transfer specifying recipient address

2. Registry Agent verifies caller is current owner
3. Ownership record atomically updated to new owner
4. Transfer event emitted for audit trail

Transfer does not reset expiration dates; the new owner inherits the remaining registration period.

5.3.2 Operator Delegation

UCNS supports operator delegation, allowing domain owners to authorize other addresses to manage domains on their behalf. This enables:

- Organizational domain management where multiple staff need access
- Smart contract based automated domain management
- Third party services operating with explicit user authorization

Operators can perform most domain management functions but cannot transfer ownership or revoke their own operator status.

5.4 Resolution Services

5.4.1 Address Resolution

The primary resolution function maps domain names to Ethereum addresses. Applications query the Resolver Agent with a domain name and receive the associated address, enabling human readable identifiers in cryptocurrency transactions and smart contract interactions.

5.4.2 Extended Metadata

Beyond address resolution, UCNS supports rich profile metadata:

Table 4: Supported Resolution Record Types

Record Type	**Data Type**	**Use Case**
ETH Address	address	Payment receiving, identity
Email	string	Contact information
Avatar	string (URL)	Profile image display
Description	string	Bio or description
Website	string (URL)	Personal/organization website
Twitter	string (handle)	Social media link
GitHub	string (username)	Developer profile

5.4.3 Resolution Query Interface

Applications query resolution data through standardized functions:

```
// Query primary address
function resolve(string memory domainName)
    public view returns (address);

// Query specific text record
function getTextRecord(string memory domainName, string memory key)
    public view returns (string memory);

// Query all records (gas intensive, off chain recommended)
function getAllRecords(string memory domainName)
    public view returns (ResolverRecord memory);
```

Listing 4: Resolution Query Interface

5.5 Dynamic Pricing

5.5.1 Pricing Model

UCNS implements tiered pricing based on domain length, reflecting market dynamics where shorter domains command premium prices:

Table 5: Domain Pricing Tiers (Example Configuration)

Domain Length	Base Price	Per Year
1 character	1.00 MATIC	+0.50 MATIC/year
2-3 characters	0.50 MATIC	+0.25 MATIC/year
4-5 characters	0.10 MATIC	+0.05 MATIC/year
6+ characters	0.05 MATIC	+0.02 MATIC/year

5.5.2 Cost Calculation Formula

The total registration cost combines base price and duration-based pricing:

$$Cost_{total} = Price_{base}(length) + (Years \times Price_{annual}(length)) \quad (1)$$

Example calculations:

- 3-character domain for 5 years: $0.50 + (5 \times 0.25) = 1.75$ MATIC
- 8-character domain for 2 years: $0.05 + (2 \times 0.02) = 0.09$ MATIC

5.5.3 Price Discovery

Users can preview costs before registration through view functions that simulate the Pricing Agent's calculation without requiring transactions. The web interface provides real time price updates as users adjust domain names and registration periods.

5.6 Web Interface

5.6.1 Interface Architecture

The UCNS web interface (index.php) provides a comprehensive user experience built on:

- **Frontend:** HTML5, CSS3, Bootstrap 5 for responsive design
- **Web3 Integration:** Ethers.js for blockchain interaction
- **Wallet Connection:** MetaMask integration for transaction signing
- **Backend:** PHP for server side rendering and session management

5.6.2 Key Interface Features

1. **Domain Search:** Real time availability checking with instant feedback
2. **Registration Form:** Guided registration with automatic cost calculation
3. **WHOIS Lookup:** Query ownership and registration details for any domain
4. **Domain Management Dashboard:** View and manage user's registered domains
5. **Record Editor:** Update resolution records with form validation
6. **Transaction History:** View past registration and update transactions

5.6.3 Wallet Integration Flow

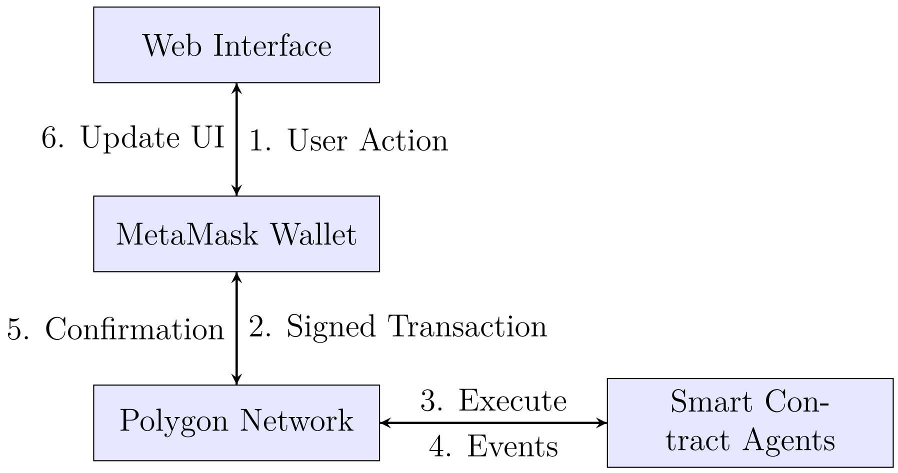

Figure 4: Web Interface Transaction Flow

6 Non Functional Requirements

6.1 Security Requirements

6.1.1 Access Control

UCNS implements multi layered access control:

- **Ownership Verification:** All ownership dependent operations verify caller authority through cryptographic signature validation
- **Authorization Gates:** Agents implement modifiers that reject unauthorized access attempts
- **Temporal Validity:** Expired domains cannot be modified; operations check expiration status
- **Operator Boundaries:** Delegated operators have limited permissions, cannot transfer ownership

6.1.2 Input Validation

All external inputs undergo rigorous validation:

- Domain names validated against naming rules before processing
- Payment amounts verified to meet or exceed calculated costs
- Address parameters checked for zero address to prevent accidental burns
- String inputs sanitized and length bounded to prevent DoS

6.1.3 Reentrancy Protection

While UCNS does not involve complex Ether transfers to external contracts, defensive patterns are implemented:

- State updates occur before external calls where applicable
- Checks Effects Interactions pattern followed in payment processing
- OpenZeppelin's ReentrancyGuard used on payment handling functions

6.1.4 Immutability and Auditability

- All state changes emit events creating immutable audit trail
- Contract source code verified on PolygonScan for public inspection
- Ownership history reconstructible from blockchain event logs
- No owner backdoors or emergency pause mechanisms (by design choice)

6.2 Performance Requirements

6.2.1 Transaction Latency

Performance characteristics depend on Polygon network conditions:

Table 6: Expected Performance Metrics

Operation	Target Latency	Gas Cost
Domain Registration	2-5 seconds	~150,000 gas
Record Update	2-5 seconds	~50,000 gas
Ownership Transfer	2-5 seconds	~80,000 gas
Availability Query (view)	<100ms	0 gas

6.2.2 Scalability

UCNS scales linearly with domain registrations:

- Each domain occupies fixed storage slots, preventing state bloat
- Query operations (view functions) run off chain with constant time complexity
- No iterative operations over unbounded data structures
- Event emission supports efficient external indexing for domain search

6.2.3 Throughput

System throughput is bounded by Polygon network capacity:

- Polygon mainnet: ~65,000 transactions per second theoretical maximum
- UCNS operations are standard contract calls, no special throughput constraints
- Concurrent registrations limited only by block gas limits and network congestion

6.3 Reliability Requirements

6.3.1 Availability

System availability depends on:

- **Polygon Network Uptime:** Target 99.9% as per Polygon's historical performance
- **RPC Provider Reliability:** Critical dependency on RPC endpoint availability
- **Web Interface:** Separate from core system; frontend unavailability does not affect on chain operations

6.3.2 Data Integrity

- All state stored on blockchain with cryptographic integrity guarantees
- Consensus mechanism ensures data consistency across network nodes
- No off chain data dependencies for critical operations
- Immutable ownership records prevent retroactive tampering

6.3.3 Fault Tolerance

- Decentralized blockchain infrastructure eliminates single points of failure
- Transaction atomicity ensures partial state updates cannot occur
- Failed transactions revert completely without corrupting state

6.4 Usability Requirements

6.4.1 User Interface

- Responsive design supporting desktop and mobile browsers
- Wallet connection process guided with clear error messaging
- Real time feedback during transaction processing
- Domain management dashboard organized by user's registered domains

6.4.2 Error Handling

- User friendly error messages replacing technical revert strings where possible
- Transaction failure reasons displayed with actionable guidance
- Network connectivity issues detected and reported clearly
- Wallet related errors distinguished from contract level errors

6.4.3 Documentation

- Comprehensive README with architecture documentation
- Smart contract interfaces documented with NatSpec comments
- Web interface includes contextual help tooltips
- Public GitHub repository with contribution guidelines

6.5 Maintainability Requirements

6.5.1 Code Quality

- Solidity code follows style guide conventions
- Clear separation of concerns across agent contracts
- Extensive inline comments explaining decision logic
- Modular function design with single responsibility principle

6.5.2 Testing

- Unit tests for individual agent functions
- Integration tests for inter agent communication flows
- Edge case testing for boundary conditions
- Gas optimization analysis

6.5.3 Upgradeability Considerations

While deployed contracts are immutable, future versions could:

- Deploy new agent contracts with improved logic
- Implement migration mechanisms for domain data
- Use proxy patterns for upgradeable components (not current implementation)
- Maintain backward compatibility with existing domains

7 Operating Environment and Constraints

7.1 Deployment Environment

7.1.1 Blockchain Platform

UCNS operates on Polygon PoS Mainnet with the following characteristics:

Table 7: Polygon Network Specifications

Parameter	Value
Network Name	Polygon Mainnet
Chain ID	137
Native Currency	MATIC
Block Time	~2 seconds
Gas Price	Dynamic, typically 30-100 Gwei
RPC Endpoint	https://polygon_rpc.com
Block Explorer	https://polygonscan.com

7.1.2 Smart Contract Deployment

All three agent contracts are deployed and verified on Polygon Mainnet:

Table 8: Deployed Contract Addresses

Contract	Address
Registry Agent	0xc9eD4B38E29C64d37cb83819D5eEcFD34EFdce0C
Resolver Agent	0x2De897131ee8AC0538585887989E2314034F0b71
Pricing Agent	0x50F50124Ee00002379142cff115b0550240898B3

7.1.3 Web Interface Hosting

The frontend is hosted at: `https://ruzbahani.com/myprojects/ucns/`

Hosting infrastructure:

- Web server: Apache/Nginx with PHP 8.0+
- SSL/TLS encryption for secure communication
- Static asset delivery via CDN for performance
- Responsive to desktop and mobile browsers

7.2 Technical Constraints

7.2.1 Blockchain Limitations

1. **Immutability:** Deployed smart contracts cannot be modified. Bug fixes require new contract deployments and potential migration mechanisms.
2. **Gas Costs:** All state changing operations require MATIC payment for gas. Complex operations are cost prohibitive.
3. **Block Gas Limit:** Individual transactions cannot exceed Polygon's block gas limit (~30 million gas per block).
4. **Computation Limits:** Smart contracts cannot perform unbounded loops or recursion due to gas constraints.
5. **Storage Costs:** On chain storage is expensive; extensive metadata requires careful optimization.
6. **Finality Time:** Transactions require block confirmation (~2 seconds) before effects are visible.

7.2.2 Solidity Language Constraints

1. **String Manipulation:** Limited native support for string operations; custom validation logic required.
2. **Floating Point:** No native floating point arithmetic; fixed point or integer math only.

3. **External Calls:** Cannot make HTTP requests or access off chain data without oracles.
4. **Randomness:** No secure source of randomness on chain without specialized oracle services.

7.2.3 Integration Constraints

1. **Wallet Dependency:** Users must have MetaMask or compatible Web3 wallet installed.
2. **MATIC Requirement:** Users need MATIC tokens for gas fees to interact with contracts.
3. **RPC Reliability:** System functionality depends on RPC endpoint availability.
4. **Browser Compatibility:** Web3 features require modern browsers with JavaScript enabled.

7.3 Operational Constraints

7.3.1 Economic Constraints

- Domain registration costs depend on MATIC price volatility
- Gas prices fluctuate based on network congestion
- Initial contract deployment incurred one time costs
- Web hosting requires ongoing maintenance fees

7.3.2 Regulatory Constraints

- Blockchain operations subject to evolving cryptocurrency regulations
- Domain names must not facilitate illegal activity
- System does not implement KYC/AML mechanisms (decentralized nature)
- Intellectual property concerns regarding trademarked names

7.3.3 Time Constraints

- Academic project timeline limits scope of features
- Testing period constrained by course schedule
- Iteration cycles limited by contract deployment immutability

8 Assumptions

The UCNS system design and implementation are based on the following assumptions:

8.1 User Assumptions

1. **Wallet Proficiency:** Users possess basic knowledge of cryptocurrency wallets and can install and configure MetaMask.
2. **MATIC Availability:** Users have access to MATIC tokens through exchanges or faucets to pay for gas fees and domain registration.
3. **Key Management:** Users maintain secure control of their private keys and understand the implications of key loss (irreversible loss of domain access).
4. **Network Selection:** Users can manually configure MetaMask to connect to Polygon Mainnet if not already configured.
5. **Transaction Understanding:** Users understand that blockchain transactions are irreversible and require careful verification before signing.

8.2 Technical Assumptions

1. **Network Stability:** Polygon PoS network maintains high uptime and reasonable gas prices for the foreseeable future.
2. **RPC Availability:** Public RPC endpoints remain accessible for contract interaction and state queries.
3. **Browser Compatibility:** Target users access the system through modern browsers (Chrome, Firefox, Edge, Safari) with Web3 support.
4. **Smart Contract Integrity:** Deployed contracts remain secure against newly discovered Solidity vulnerabilities.
5. **No Hard Forks:** Polygon network does not undergo consensus breaking hard forks that would invalidate deployed contracts.

8.3 Operational Assumptions

1. **Domain Naming:** Users voluntarily comply with naming rules and avoid registering offensive or trademarked names.
2. **Renewal Responsibility:** Domain owners monitor expiration dates and renew domains proactively; no automated reminder system exists.
3. **Metadata Accuracy:** Users provide accurate resolution records; the system does not verify email addresses or URLs.
4. **No Censorship:** No entity can prevent users from registering domains or updating records (inherent blockchain property).
5. **Economic Rationality:** Pricing tiers reflect reasonable market expectations; extreme MATIC price changes may require Pricing Agent updates.

8.4 Scope Assumptions

1. **Single Top Level Domain:** All domains end with ".ucns"; multi-TLD support is out of scope.
2. **No Subdomain Delegation:** Domain owners cannot create subdomains; each registration is independent.
3. **Limited Governance:** System parameters (pricing, reserved names) are controlled by contract owner; no DAO governance implemented.
4. **Metadata Storage:** On chain metadata storage is acceptable for the academic project scope; off chain storage integration not required.
5. **Domain Trading:** Peer to peer domain transfers occur through direct ownership transfer; no built in marketplace or escrow.

9 Limitations

While UCNS provides a functional and architecturally sound naming service, several limitations exist due to scope constraints, technical trade offs, and academic project boundaries:

9.1 Feature Limitations

1. **No Decentralized Governance:** System parameters (pricing tiers, reserved names) are controlled by contract owner rather than community voting. Implementing DAO governance would require additional governance tokens and voting mechanisms beyond current scope.
2. **Manual Renewal Only:** Domains must be manually renewed before expiration. The system lacks automated renewal through subscription mechanisms or grace periods, requiring active monitoring by domain owners.
3. **No Marketplace Functionality:** While ownership transfer is supported, the system does not include built in domain marketplace features such as listing, bidding, or escrow services. Domain trading must occur through external platforms or manual peer to peer transfers.
4. **Single Level Domains:** UCNS supports only top level ".ucns" domains. Subdomain delegation (e.g., "sub.example.ucns") is not implemented, limiting hierarchical naming structures.
5. **Limited Record Types:** The system supports a fixed set of resolution record types. Extensible or custom record types would require contract upgrades or additional resolver implementations.
6. **No Reverse Resolution:** The system does not support reverse lookups (address to domain name), which would require additional indexing and storage mechanisms.

9.2 Technical Limitations

1. **Contract Immutability:** Deployed smart contracts cannot be upgraded without migration mechanisms. Bug fixes or feature additions require new contract deployments and potential data migration.
2. **Gas Cost Optimization:** While reasonably optimized, certain operations (particularly bulk domain management) could benefit from further gas optimization techniques not implemented in the current version.
3. **Off-Chain Metadata:** All metadata is stored on chain, increasing costs. Integration with decentralized storage (IPFS, Arweave) for large metadata would reduce gas costs but add complexity.
4. **Query Performance:** Fetching all domains for a specific owner requires iterating through events or maintaining off chain indexes, as on chain enumeration would be gas prohibitive.
5. **Name Squatting Protection:** The system lacks mechanisms to prevent domain squatting or trademark abuse beyond pricing tiers and reserved names.
6. **Internationalization:** Domain names are limited to ASCII character set; international domain names (IDN) with Unicode characters are not supported.

9.3 Integration Limitations

1. **Single Chain Operation:** UCNS operates exclusively on Polygon. Cross chain resolution or multi chain synchronization is not implemented.
2. **DNS Integration:** The system does not bridge to traditional DNS, requiring separate resolution mechanisms for Web2 and Web3 contexts.
3. **Wallet Dependency:** System interaction requires MetaMask or compatible Web3 wallet; no alternative authentication mechanisms exist.
4. **Browser Extension:** Unlike ENS, UCNS does not have a browser extension for automatic domain resolution in web browsers.
5. **Mobile Application:** No dedicated mobile application exists; mobile users must use mobile web browsers with wallet support.

9.4 Economic Limitations

1. **Price Volatility:** Domain costs in MATIC are subject to cryptocurrency price volatility, potentially making domains expensive if MATIC appreciates significantly.
2. **Gas Fee Dependency:** All operations incur gas fees, making the system less accessible during network congestion or to users with limited MATIC.
3. **No Pricing Oracles:** Pricing is static relative to MATIC; the system does not integrate price oracles to maintain stable USD denominated pricing.

9.5 Operational Limitations

1. **Limited Support:** As an academic project, no formal customer support or SLA guarantees exist.
2. **Testing Coverage:** While functional testing was performed, comprehensive security audits by third party firms have not been conducted.
3. **Documentation Gaps:** Some advanced use cases and integration patterns may lack detailed documentation.
4. **Community Ecosystem:** UCNS lacks the extensive tooling, integrations, and community support available to established naming systems like ENS.

9.6 Future Enhancement Opportunities

These limitations present opportunities for future development phases:

- Implement upgradeable proxy patterns for contract evolution
- Develop decentralized governance mechanisms with voting tokens
- Create automated renewal and subscription services
- Build domain marketplace with escrow and bidding
- Add subdomain delegation capabilities
- Integrate with decentralized storage for large metadata
- Develop browser extensions and mobile applications
- Implement cross-chain bridges for multi chain resolution
- Add reputation systems and squatting prevention mechanisms

10 Conclusion

10.1 Summary of Achievements

The University of Calgary Name Service successfully demonstrates the application of agent-based software engineering principles to build a production grade decentralized naming system on the Polygon blockchain. Through the decomposition of naming service functionality into three specialized autonomous agents Registry, Resolver, and Pricing the project illustrates how complex distributed systems can be architected using agent oriented methodologies.

Key achievements include:

Architectural Clarity: The system exhibits clear separation of concerns with each agent maintaining distinct responsibilities, independent state, and autonomous decision-making capabilities. This modularity facilitates reasoning about system behavior and enables independent verification of agent properties.

Practical Implementation: Moving beyond theoretical agent design, UCNS provides a fully functional system deployed on Polygon mainnet with verified smart contracts and a user friendly web interface. The implementation demonstrates that agent oriented designs are not merely academic exercises but viable approaches for real world blockchain applications.

Agent Autonomy: Each agent operates independently without centralized control, making decisions based on internal logic and explicit inter agent communication. This autonomy reflects core agent paradigm principles while leveraging blockchain's trustless execution guarantees.

Decentralized Coordination: The system showcases how autonomous agents coordinate through message passing and structured interfaces to deliver coherent system level behavior without requiring centralized orchestration.

10.2 Agent-Based Design Benefits

The agent-based approach in UCNS provides several concrete advantages:

Modularity: Clear agent boundaries enable independent development, testing, and potential upgrading of system components. The Pricing Agent, for example, can be replaced with minimal impact on Registry or Resolver agents.

Understandability: Decomposing the naming service into specialized agents with focused responsibilities makes the system easier to comprehend than monolithic alternatives. Each agent's purpose and behavior can be understood in isolation.

Verifiability: Agent level properties (e.g., "Registry Agent never assigns ownership to zero address" or "Resolver Agent always verifies ownership before updates") can be formally specified and verified independently.

Extensibility: New agents can be added to the system without modifying existing agents. Future marketplace agents or governance agents would integrate through defined interfaces.

Alignment with Blockchain: Smart contracts naturally exhibit agent characteristics (autonomy, deterministic behavior, message based communication), making agent oriented design particularly suitable for blockchain systems.

10.3 Educational Value

For the SENG 696 course context, UCNS demonstrates:

- Practical application of GAIA methodology to real world system design
- Translation of agent oriented analysis into executable code
- Benefits and trade offs of agent-based decomposition in distributed systems
- Coordination patterns among autonomous agents in trustless environments
- Integration challenges when bridging agent-based backends with traditional UIs

10.4 Contributions to Blockchain Naming

While UCNS is scoped as an academic project, it contributes to the broader blockchain naming ecosystem by:

- Providing an open source reference implementation of agent-based naming services
- Demonstrating alternative architectural approaches beyond monolithic contract designs
- Offering a lightweight naming system suitable for educational purposes and experimentation
- Illustrating how Layer 2 solutions (Polygon) enable practical blockchain applications with low costs

10.5 Reflection on Limitations

The documented limitations of UCNS are primarily scope driven rather than fundamental architectural flaws. The agent-based design actually facilitates addressing many limitations:

- Subdomain support could be added through a new Subdomain Agent
- Marketplace functionality could integrate via a dedicated Trading Agent
- Governance mechanisms could be implemented as a Governance Agent
- Storage optimization could leverage a Storage Agent interfacing with IPFS

This extensibility potential validates the agent-based approach's suitability for evolving system requirements.

10.6 Future Research Directions

UCNS opens several avenues for future research and development:

Formal Verification: Apply formal methods to prove agent properties and coordination correctness.

Agent Learning: Explore adaptive pricing mechanisms where the Pricing Agent learns optimal pricing from market behavior.

Multi Agent Protocols: Investigate more sophisticated coordination protocols beyond simple function calls, potentially implementing FIPA ACL style message passing.

Cross Chain Agents: Research agent architectures for cross chain name resolution and ownership synchronization.

Performance Optimization: Study gas optimization techniques specific to multi agent smart contract systems.

Security Analysis: Conduct comprehensive security audits examining attack vectors in agent communication patterns.

10.7 Final Remarks

The University of Calgary Name Service demonstrates that agent-based software engineering principles, when applied thoughtfully to blockchain systems, yield architectures that are modular, understandable, and extensible. The project successfully bridges theoretical agent oriented design concepts from SENG 696 with practical blockchain implementation, providing both a functional decentralized application and an educational artifact for studying multi agent systems in trustless environments.

The system's three autonomous agents Registry, Resolver, and Pricing coordinate seamlessly to provide comprehensive naming services while maintaining clear boundaries and independent operation. This architectural approach not only satisfies academic objectives but also produces a viable alternative to monolithic naming system designs, potentially influencing future blockchain infrastructure development.

As blockchain technology continues to evolve and mature, agent-based design methodologies offer promising pathways for building complex, decentralized systems that are both theoretically sound and practically effective. UCNS represents one step in exploring this convergence of agent oriented software engineering and blockchain infrastructure.

Appendix A: Glossary

Agent An autonomous software entity that perceives its environment, makes decisions based on internal logic, and acts to achieve goals.

Blockchain A distributed ledger technology providing immutable, transparent record keeping through cryptographic consensus.

dApp (Decentralized Application) A software application that runs on a blockchain rather than centralized servers.

ENS (Ethereum Name Service) The original blockchain naming system mapping names to Ethereum addresses.

EVM (Ethereum Virtual Machine) The runtime environment for executing smart contracts.

Gas Computational unit measuring execution cost of blockchain operations.

Hash Cryptographic fingerprint uniquely identifying data (e.g., domain names).

MATIC Native cryptocurrency of the Polygon network used for transaction fees.

MetaMask Popular browser extension wallet for interacting with Ethereum and compatible blockchains.

Polygon Layer 2 scaling solution for Ethereum providing fast, low cost transactions.

Resolver Agent or system component translating domain names to addresses or metadata.

Smart Contract Self executing code deployed on blockchain that runs deterministically when conditions are met.

Solidity Programming language for writing Ethereum smart contracts.

Web3 Paradigm for decentralized internet applications leveraging blockchain technology.

Appendix B: Contract Addresses

Table 9: Deployed Contract Information

Contract	Address	Verified
Registry Agent	0xc9eD4B38E29C64d37cb83819D5eEcFD34EFdce0C	
Resolver Agent	0x2De897131ee8AC0538585887989E2314034F0b71	
Pricing Agent	0x50F50124Ee00002379142cff115b0550240898B3	

All contracts are deployed on Polygon Mainnet (Chain ID: 137) and verified on Polygon-Scan.

Appendix C: Web Interface URL

Live Demo: `https://ruzbahani.com/myprojects/ucns/`

Appendix D: References

1. Wooldridge, M. (2009). *An Introduction to MultiAgent Systems.* John Wiley & Sons.

2. Jennings, N. R., Sycara, K., & Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous agents and multi-agent systems*, 1(1), 7-38.

3. Buterin, V. (2014). *Ethereum: A next-generation smart contract and decentralized application platform.* Ethereum White Paper.

4. Johnson, D., Menezes, A., & Vanstone, S. (2001). The elliptic curve digital signature algorithm (ECDSA). *International journal of information security*, 1(1), 36-63.

5. Zamfir, V. (2015). *Introducing Casper "the Friendly Ghost".* Ethereum Blog.

6. Polygon Technology. (2021). *Polygon PoS Chain Documentation.* Retrieved from https://docs.polygon.technology/

7. OpenZeppelin. (2023). *OpenZeppelin Contracts Documentation.* Retrieved from https://docs.openzeppelin.com/

8. Far, B. H. (2024). *SENG 696 Course Materials - Agent-Based Software Engineering.* University of Calgary.

9. ENS Labs. (2023). *ENS Documentation.* Retrieved from https://docs.ens.domains/

10. Antonopoulos, A. M., & Wood, G. (2018). *Mastering Ethereum: Building smart contracts and DApps.* O'Reilly Media.