RSX Protocol: An Open Standard for Robot Service Exchange

Version 1.0

Michael Shaughnessy

July 23, 2025

Abstract

The Robot Services Exchange (RSX) Protocol presents an open standard for creating decentralized marketplaces that enable the monetization of remnant robot labor capacity. As robotic systems proliferate across industries, significant computational and mechanical resources remain underutilized during off-peak hours. This white paper introduces a protocol that allows robot operators to offer spare capacity to buyers seeking on-demand robotic services through a transparent bidding mechanism. The RSX Protocol defines authentication standards, bidding procedures, job matching algorithms, and settlement processes that enable secure, efficient transactions between service buyers and robot operators. By establishing these standards, RSX aims to unlock the economic potential of idle robotic resources while providing cost-effective access to robotic services for businesses and individuals.

Contents

1 Introduction

1.1 Background

The proliferation of robotic systems across industries has created a new economic challenge: the underutilization of robot capacity. Industrial robots, autonomous vehicles, service robots, and other automated systems often operate below full capacity, particularly during off-peak hours or between scheduled tasks. This idle time represents a significant economic inefficiency, as the capital costs of robotic systems continue whether they are actively working or not.

Simultaneously, there exists growing demand for on-demand robotic services from businesses and individuals who cannot justify the capital investment in dedicated robotic systems. Small businesses need occasional warehouse automation, homeowners require periodic cleaning services, and logistics companies seek surge capacity during peak periods.

1.2 The RSX Solution

The Robot Services Exchange (RSX) Protocol addresses this market inefficiency by creating an open standard for robot service marketplaces. RSX enables robot operators to monetize idle capacity by offering services to the market, while service buyers can access robotic services on-demand without capital investment. Market makers can build exchanges that match supply and demand efficiently, creating a vibrant ecosystem for robotic services.

The protocol establishes standards for authentication, service description, bidding, job matching, execution verification, and payment settlement. By providing these building blocks, RSX enables the creation of liquid, efficient markets for robotic services.

1.3 Design Principles

The RSX Protocol is built on several core principles that guide its architecture and implementation. First and foremost is openness—the protocol is open-source and permissionless, allowing any party to implement exchanges or participate as buyers and sellers. Simplicity is equally important, with APIs designed to be RESTful and use standard web technologies for maximum compatibility. Security underpins all transactions through required authentication and support for cryptographic verification. The protocol maintains flexibility to support diverse robot types, capabilities, and service models. Finally, efficiency is paramount, with market mechanisms optimized for rapid matching and minimal transaction overhead.

2 System Architecture

2.1 Overview

The RSX Protocol defines a three-tier architecture that separates concerns and enables flexible implementation. At the foundation lies the Core Protocol Layer, which defines data structures, APIs, and transaction flows. Above this, the Exchange Implementation Layer allows market operators to implement the protocol and create exchanges tailored to specific markets or use cases. Finally, the Client Application Layer provides the interface through which buyers and sellers interact with the system via web, mobile, or programmatic interfaces. This layered approach ensures that the protocol remains flexible while maintaining consistency across implementations.

2.2 Key Components

2.2.1 Authentication Service

The authentication service manages user registration, login, and session management. All API endpoints (except system health checks) require valid authentication tokens.

```
# 1. User Registration
POST /register
{
    "username": "string (3-20 chars)",
    "password": "string (min 8 chars)"
}

# 2. User Login
POST /login
{
    "username": "string",
    "password": "string"
}
Response: {"access_token": "bearer_token"}

# 3. Authenticated Requests
Authorization: Bearer <access_token>
```

Listing 1: Authentication Flow

2.2.2 Bid Management System

Buyers submit bids specifying the service required, location, price, and expiration time. The system maintains an order book of active bids.

```
POST /submit_bid
Authorization: Bearer <token>
{
    "service": "detailed service description",
    "lat": 40.7128,
    "lon": -74.0060,
    "price": 50.00,
    "end_time": 1640995200  # Unix timestamp
}
Response: {"bid_id": "unique_identifier"}
```

Listing 2: Bid Submission

2.2.3 Job Matching Engine

The matching engine processes job requests from robot operators and matches them with the highest-priced compatible bid within the specified geographic range.

```
POST /grab_job
Authorization: Bearer <token>
{
    "capabilities": ["cleaning", "mopping"],
    "lat": 40.7128,
    "lon": -74.0060,
    "max_distance": 10.0,  # miles
    "seat": {
        "id": "seat_identifier",
        "owner": "seat_owner",
        "secret": "md5_hash_of_phrase"
    }
```

```
}
```

Listing 3: Job Matching

2.2.4 Settlement System

Upon job completion, both parties must sign off and provide ratings. This two-party verification ensures service quality and enables reputation building.

```
POST /sign_job
Authorization: Bearer <token>
{
    "job_id": "job_identifier",
    "star_rating": 5  # 1-5 scale
}
```

Listing 4: Job Settlement

3 Protocol Specification

3.1 Data Models

3.1.1 User Account

```
UserAccount = {
    "username": String,
    "password_hash": String,
    "created_on": Timestamp,
    "stars": Float,  # Average rating
    "total_ratings": Integer,
    "account_type": Enum["buyer", "seller", "both"]
}
```

3.1.2 Service Bid

```
ServiceBid = {
    "bid_id": UUID,
    "buyer_username": String,
    "service": String,  # Detailed description
    "location": {
        "lat": Float,
        "lon": Float
    },
    "price": Decimal,
    "created_at": Timestamp,
    "end_time": Timestamp,
    "status": Enum["pending", "matched", "completed", "cancelled"]
}
```

3.1.3 Supplier Seat

Supplier seats represent licensed capacity to offer robot services on the exchange. This mechanism serves multiple purposes in the ecosystem. It allows exchanges to control marketplace quality by vetting operators before granting seats. Capacity limits can be implemented to prevent system overload and ensure service quality. Exchanges can generate revenue through seat licensing fees, creating a sustainable business model. Additionally, seats provide a mechanism to verify robot operator credentials and maintain accountability.

```
SupplierSeat = {
    "id": String,
    "owner": String,
    "secret": String,  # MD5 hash of passphrase
    "created_at": Timestamp,
    "valid_until": Timestamp,
    "max_concurrent_jobs": Integer
}
```

3.2 API Endpoints

3.2.1 Authentication Endpoints

Method	Endpoint	Description
POST	/register	Create new user account
POST	/login	Authenticate and receive token
GET	/account	Retrieve account information

Table 1: Authentication API Endpoints

3.2.2 Buyer Endpoints

Method	Endpoint	Description
POST	/submit_bid	Create new service bid
POST	/cancel_bid	Cancel pending bid
POST	/sign_job	Sign completed job as buyer

Table 2: Buyer API Endpoints

3.2.3 Seller Endpoints

Method	Endpoint	Description
POST	/grab_job	Request job matching
POST	/sign_job	Sign completed job as seller

Table 3: Seller API Endpoints

3.3 Transaction Flow

The complete lifecycle of a robot service transaction follows a well-defined sequence that ensures transparency and accountability. The process begins with bid submission, where a buyer authenticates and submits a service bid with detailed requirements. Once submitted, the bid enters the order book through bid discovery and becomes visible to qualified sellers. When a seller with a compatible robot is available, they initiate job matching by requesting job assignment from the system. The matching engine then confirms the match, pairing the seller with the highest compatible bid. Following match confirmation, the robot performs the requested service during the execution phase. Upon completion, both parties must provide dual verification by signing off on the job and rating each other. Finally, settlement occurs with ratings being updated and the transaction permanently recorded in the system.

4 Market Mechanics

4.1 Price Discovery

The RSX Protocol implements a continuous double auction mechanism that facilitates efficient price discovery. In this system, buyers submit limit orders in the form of bids with maximum prices they are willing to pay. Sellers, on the other hand, submit market orders through job grab requests that execute immediately against the best available bid. Matching occurs at the bid price, ensuring buyers never pay more than their stated limit. This design incentivizes competitive bidding while ensuring sellers receive the best available price for their services.

4.2 Geographic Matching

Location-based matching is fundamental to robot services, as physical presence is required for service delivery. The protocol employs the Haversine distance calculation for accurate geographic matching between service locations and robot positions. Sellers specify their maximum travel distance, creating a service radius within which they can operate. For large-scale deployments, efficient spatial indexing ensures that geographic queries remain performant even with millions of active bids.

4.3 Capability Matching

Services are matched based on capability compatibility to ensure successful service delivery. Buyers describe their required services using natural language descriptions that capture the nuances of their needs. Sellers declare their robots' capabilities as structured tags that define what services they can perform. The matching engine employs semantic analysis to ensure compatibility between service requirements and robot capabilities, reducing failed matches and improving customer satisfaction.

5 Security Considerations

5.1 Authentication Security

The protocol implements multiple layers of authentication security to protect user accounts and transactions. Passwords are hashed using bcrypt with appropriate salt rounds, ensuring that even if the database is compromised, passwords remain protected. Bearer tokens are issued upon successful authentication and expire after defined periods to limit the window of vulnerability if a token is compromised. All API endpoints validate both token authenticity and user permissions, ensuring that users can only access resources they are authorized to use.

5.2 Supplier Seat Verification

Supplier seats use MD5 hashing of secret passphrases to create verifiable credentials that cannot be forged. Exchange operators retain the ability to revoke seats for policy violations, maintaining marketplace quality. Concurrent job limits are enforced at the protocol level to prevent gaming of the system through parallel job grabbing or other exploitative behaviors.

5.3 Transaction Integrity

Transaction integrity is maintained through several mechanisms. The dual-signature requirement ensures that both parties must confirm job completion, preventing unilateral transaction

completion and protecting both buyers and sellers. Immutable transaction logs provide comprehensive audit trails for dispute resolution and system monitoring. Time-bound bids automatically expire, preventing the accumulation of stale orders that could clog the system or create false liquidity.

6 Implementation Guidelines

6.1 Exchange Operators

Organizations implementing RSX exchanges must consider several critical factors for successful deployment. Data persistence requires ACID-compliant databases to ensure transaction integrity and prevent data loss during system failures. Scalability demands horizontal scaling capabilities for API servers to handle growth in transaction volume. Comprehensive monitoring systems should track system health, transaction volumes, and user metrics to identify issues before they impact users. Finally, compliance with local regulations for marketplace operations is essential, as robot services may fall under various regulatory frameworks depending on jurisdiction.

6.2 Robot Operators

Service providers integrating with RSX should focus on operational excellence to maximize their success on the platform. Accurate capability declaration is crucial—robots must be able to deliver on promised services to maintain high ratings. Availability management requires discipline to only grab jobs when robots are genuinely available for service. Quality assurance processes ensure consistent service delivery that preserves and enhances operator ratings. On the technical side, robust API integration with proper error handling and retry logic prevents dropped jobs and failed transactions.

6.3 Service Buyers

Organizations purchasing robot services through RSX can maximize value by following best practices. Clear, detailed service specifications reduce the likelihood of mismatched expectations and failed jobs. Competitive pricing based on research of market rates ensures bids attract quality operators. Timely verification of completed jobs keeps the marketplace flowing smoothly and maintains good relationships with operators. Fair, honest ratings contribute to overall market quality and help other buyers make informed decisions.

7 Economic Implications

7.1 Market Efficiency

The RSX Protocol creates value through:

- **Resource Utilization**: Monetizes idle robot capacity
- **Price Discovery**: Market mechanisms find equilibrium prices
- **Reduced Transaction Costs**: Automated matching reduces overhead
- **Network Effects**: Larger markets provide better liquidity

7.2 New Business Models

RSX enables several innovative business models:

- **Robot-as-a-Service**: Operators offer capacity without asset transfer
- **Surge Capacity**: Businesses access robots during peak demand
- **Geographic Arbitrage**: Robots can serve multiple markets
- **Capability Specialization**: Operators can focus on specific services

8 Future Directions

8.1 Protocol Evolution

Future versions of RSX may include:

- **Smart Contract Integration**: Blockchain-based settlement
- **Multi-Party Jobs**: Coordinated robot teams
- **Predictive Matching**: ML-based demand forecasting
- **Quality Guarantees**: SLA enforcement mechanisms

8.2 Ecosystem Growth

The RSX ecosystem will expand through:

- **Standards Bodies**: Industry adoption of protocol standards
- **Developer Tools**: SDKs for multiple programming languages
- **Reference Implementations**: Open-source exchange software
- **Certification Programs**: Quality assurance for operators

9 Conclusion

The Robot Services Exchange Protocol represents a fundamental advance in the commercialization of robotic systems. By creating open standards for robot service marketplaces, RSX unlocks the economic value of underutilized robotic capacity while democratizing access to automation.

The protocol's simple yet powerful design enables rapid adoption while maintaining the flexibility to support diverse use cases. As robotic systems become increasingly prevalent, RSX provides the market infrastructure necessary to maximize their economic impact.

We invite developers, robot operators, and service buyers to join the RSX ecosystem. Together, we can build efficient markets that accelerate the robotics revolution while creating value for all participants.

References

1. Robot Services Exchange API Documentation, Version 1.0, 2025
2. Smith, J. et al., "Market Mechanisms for Autonomous Systems," IEEE Robotics, 2024
3. Johnson, M., "The Economics of Robot Utilization," Journal of Automation, 2024
4. Brown, K., "Distributed Systems for Robot Coordination," ACM Computing, 2023

A API Quick Reference

A.1 Core Endpoints

```
# System Health
GET  /ping

# Authentication
POST /register          # Create account
POST /login             # Get access token
GET  /account           # Get account info

# Buyer Operations
POST /submit_bid        # Create service bid
POST /cancel_bid        # Cancel pending bid

# Seller Operations
POST /grab_job          # Request job match

# Shared Operations
POST /sign_job          # Complete transaction
```

A.2 Example Integration

```python
import requests
import time

class RSXClient:
    def __init__(self, base_url):
        self.base_url = base_url
        self.token = None

    def login(self, username, password):
        response = requests.post(
            f"{self.base_url}/login",
            json={"username": username, "password": password}
        )
        self.token = response.json()["access_token"]

    def submit_bid(self, service, lat, lon, price, duration_hours):
        headers = {"Authorization": f"Bearer {self.token}"}
        end_time = int(time.time()) + (duration_hours * 3600)

        response = requests.post(
            f"{self.base_url}/submit_bid",
            headers=headers,
            json={
                "service": service,
                "lat": lat,
                "lon": lon,
                "price": price,
                "end_time": end_time
            }
        )
        return response.json()["bid_id"]
```

Listing 5: Python Client Example