Cross-Language Development Example: Python Service to C# Client

Demonstrating seamless cross-language development with automatic schema generation and code generation

---

Scenario Overview

User A (Python Developer) creates a user management service in Python with automatic schema generation. **User B (C# Developer)** consumes the Python service from C# using generated client code.

This example demonstrates the platform's ability to provide seamless cross-language integration through OpenRPC schemas and automatic code generation.

User A: Python Service Development

Step 1: Create Python Service Function

File: `src/services/user_service.py`

python

```python
from platform_core import ServiceFunction, Function
from typing import Optional
from dataclasses import dataclass
from datetime import datetime

@dataclass
class CreateUserRequest:
    """Request to create a new user"""
    email: str # User's email address - must be unique
    name: str  # User's full name
    department: Optional[str] = None # Optional department assignment

@dataclass
class CreateUserResponse:
    """Response from user creation"""
    user_id: str  # Unique identifier for the created user
    success: bool  # Indicates whether the operation succeeded
    message: Optional[str] = None # Optional success/error message

@dataclass
class User:
    """User entity with all properties"""
    id: str
    email: str
    name: str
    department: Optional[str]
    created_at: datetime
    active: bool = True

@ServiceFunction("UserManagement")
class UserService:
    def __init__(self, data_store, messaging, authorizer):
        self.data_store = data_store
        self.messaging = messaging
        self.authorizer = authorizer

    @Function("createUser")
    async def create_user(self, request: CreateUserRequest) -> CreateUserResponse:
        """Create a new user account"""
        try:
            # Check for existing user
            existing_users = await self.data_store.query(
                "users",
```

```
            {"email": request.email}
        )

        if existing_users:
            return CreateUserResponse(
                user_id="",
                success=False,
                message="User with this email already exists"
            )

        # Create new user
        user_data = {
            "email": request.email,
            "name": request.name,
            "department": request.department,
            "created_at": datetime.utcnow().isoformat(),
            "active": True
        }

        user_id = await self.data_store.insert("users", user_data)

        # Send notification
        await self.messaging.send("notifications", {
            "type": "user_created",
            "user_id": user_id,
            "email": request.email
        })

        return CreateUserResponse(
            user_id=user_id,
            success=True,
            message="User created successfully"
        )

    except Exception as e:
        return CreateUserResponse(
            user_id="",
            success=False,
            message=f"Error creating user: {str(e)}"
        )

@Function("getUser")
async def get_user(self, user_id: str) -> Optional[User]:
    """Retrieve a user by ID"""
    user_data = await self.data_store.get_by_id("users", user_id)
```

```
    if not user_data:
        return None

    return User(
        id=user_data["id"],
        email=user_data["email"],
        name=user_data["name"],
        department=user_data.get("department"),
        created_at=datetime.fromisoformat(user_data["created_at"]),
        active=user_data.get("active", True)
    )

@Function("listUsers")
async def list_users(self, department: Optional[str] = None) -> list[User]:
    """List all users, optionally filtered by department"""
    query = {"department": department} if department else {}
    users_data = await self.data_store.query("users", query)

    return [
        User(
            id=user["id"],
            email=user["email"],
            name=user["name"],
            department=user.get("department"),
            created_at=datetime.fromisoformat(user["created_at"]),
            active=user.get("active", True)
        )
        for user in users_data
    ]
```

Step 2: Deploy Python Service

bash

```
# Deploy the Python service
cd python-user-service
platform-cli dev start --watch

# Output:
# 🚀 Rapid restart active - watching for changes
# 📝 Analyzing Python service functions...
# 🔄 Generating OpenRPC schema from type annotations...
# 📋 Generated schema: schemas/UserManagement.openrpc.json
# 🚀 UserManagement service deployed
# 📡 MQTT subscribed: service/UserManagement
# ✅ Rapid restart complete (4.1s)
```

Step 3: Auto-Generated OpenRPC Schema

The platform automatically generates this OpenRPC schema from the Python type annotations:

File: `schemas/UserManagement.openrpc.json` (Auto-generated)

json


```
{
 "openrpc": "1.3.0",
 "info": {
  "title": "UserManagement",
  "version": "1.0.0",
  "description": "User management service functions"
 },
 "methods": [
  {
   "name": "createUser",
   "summary": "Create a new user account",
   "params": [
    {
     "name": "request",
     "required": true,
     "schema": {
      "$ref": "#/components/schemas/CreateUserRequest"
     }
    }
   ],
   "result": {
    "name": "CreateUserResponse",
    "description": "Response from user creation",
    "schema": {
     "$ref": "#/components/schemas/CreateUserResponse"
    }
   }
  },
  {
   "name": "getUser",
   "summary": "Retrieve a user by ID",
   "params": [
    {
     "name": "user_id",
     "required": true,
     "schema": {
      "type": "string",
      "format": "uuid"
     }
    }
   ],
   "result": {
    "name": "User",
```

```
    "description": "User entity with all properties",
    "schema": {
     "oneOf": [
      {"$ref": "#/components/schemas/User"},
      {"type": "null"}
     ]
    }
   }
  },
  {
   "name": "listUsers",
   "summary": "List all users, optionally filtered by department",
   "params": [
    {
     "name": "department",
     "required": false,
     "schema": {
      "oneOf": [
       {"type": "string"},
       {"type": "null"}
      ]
     }
    }
   ],
   "result": {
    "name": "UserList",
    "description": "List of users",
    "schema": {
     "type": "array",
     "items": {
      "$ref": "#/components/schemas/User"
     }
    }
   }
  }
 ],
 "components": {
  "schemas": {
   "CreateUserRequest": {
    "type": "object",
    "title": "CreateUserRequest",
    "description": "Request to create a new user",
    "properties": {
     "email": {
      "type": "string",
      "format" "email"
```

```
      "format": "email",
      "description": "User's email address - must be unique"
     },
     "name": {
      "type": "string",
      "minLength": 1,
      "maxLength": 100,
      "description": "User's full name"
     },
     "department": {
      "oneOf": [
       {"type": "string"},
       {"type": "null"}
      ],
      "description": "Optional department assignment"
     }
    },
    "required": ["email", "name"],
    "additionalProperties": false
   },
   "CreateUserResponse": {
    "type": "object",
    "title": "CreateUserResponse",
    "description": "Response from user creation",
    "properties": {
     "user_id": {
      "type": "string",
      "format": "uuid",
      "description": "Unique identifier for the created user"
     },
     "success": {
      "type": "boolean",
      "description": "Indicates whether the operation succeeded"
     },
     "message": {
      "oneOf": [
       {"type": "string"},
       {"type": "null"}
      ],
      "description": "Optional success/error message"
     }
    },
    "required": ["user_id", "success"],
    "additionalProperties": false
   },
   "User": {
```

```
    "User": {
      "type": "object",
      "title": "User",
      "description": "User entity with all properties",
      "properties": {
        "id": {
          "type": "string",
          "format": "uuid"
        },
        "email": {
          "type": "string",
          "format": "email"
        },
        "name": {
          "type": "string"
        },
        "department": {
          "oneOf": [
            {"type": "string"},
            {"type": "null"}
          ]
        },
        "created_at": {
          "type": "string",
          "format": "date-time"
        },
        "active": {
          "type": "boolean",
          "default": true
        }
      },
      "required": ["id", "email", "name", "created_at"],
      "additionalProperties": false
    }
  }
 }
}
```

User B: C# Client Development

Step 1: Discover Available Services

```bash
# User B discovers available services
platform-cli services list

# Output:
# Available Services:
# ✅ UserManagement (v1.0.0) - User management service functions
#    📍 service/UserManagement
#    📋 Schema: http://localhost:8080/schemas/UserManagement.openrpc.json
#    🔧 Methods: createUser, getUser, listUsers
```

Step 2: Generate C# Client Code

```bash
# Create new C# project
mkdir csharp-user-client
cd csharp-user-client

# Generate C# client from Python service schema
platform-cli generate client \
 --service UserManagement \
 --language csharp \
 --output ./Generated

# Output:
# 📋 Loading schema: UserManagement.openrpc.json
# 🔄 Generating C# client code...
# 📁 Generated files:
#    - Generated/UserManagement/UserManagementClient.cs
#    - Generated/UserManagement/Models.cs
#    - Generated/UserManagement/IUserManagementClient.cs
# ✅ C# client generation complete
```

Step 3: Generated C# Client Code

File: `Generated/UserManagement/Models.cs` (Auto-generated)

csharp

```csharp
// Auto-generated from UserManagement OpenRPC schema - DO NOT EDIT
using System;
using System.ComponentModel.DataAnnotations;

namespace Generated.UserManagement
{
    /// <summary>
    /// Request to create a new user
    /// </summary>
    public class CreateUserRequest
    {
        /// <summary>
        /// User's email address - must be unique
        /// </summary>
        [Required]
        [EmailAddress]
        public string Email { get; set; }

        /// <summary>
        /// User's full name
        /// </summary>
        [Required]
        [StringLength(100, MinimumLength = 1)]
        public string Name { get; set; }

        /// <summary>
        /// Optional department assignment
        /// </summary>
        public string? Department { get; set; }
    }

    /// <summary>
    /// Response from user creation
    /// </summary>
    public class CreateUserResponse
    {
        /// <summary>
        /// Unique identifier for the created user
        /// </summary>
        [Required]
        public string UserId { get; set; }

        /// <summary>
```

```
    /// Indicates whether the operation succeeded
    /// </summary>
    [Required]
    public bool Success { get; set; }

    /// <summary>
    /// Optional success/error message
    /// </summary>
    public string? Message { get; set; }
  }

  /// <summary>
  /// User entity with all properties
  /// </summary>
  public class User
  {
    [Required]
    public string Id { get; set; }

    [Required]
    [EmailAddress]
    public string Email { get; set; }

    [Required]
    public string Name { get; set; }

    public string? Department { get; set; }

    [Required]
    public DateTime CreatedAt { get; set; }

    public bool Active { get; set; } = true;
  }
}
```

File: `Generated/UserManagement/UserManagementClient.cs` (Auto-generated)

csharp

```csharp
// Auto-generated from UserManagement OpenRPC schema - DO NOT EDIT
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Platform.Messaging;
using Microsoft.Extensions.Logging;

namespace Generated.UserManagement
{
  /// <summary>
  /// User management service functions
  /// Generated client for UserManagement v1.0.0
  /// </summary>
  public class UserManagementClient : IUserManagementClient
  {
    private readonly IMessaging _messaging;
    private readonly ILogger<UserManagementClient> _logger;
    private const string ServiceTopic = "service/UserManagement";

    public UserManagementClient(IMessaging messaging, ILogger<UserManagementClient> logger)
    {
      _messaging = messaging ?? throw new ArgumentNullException(nameof(messaging));
      _logger = logger ?? throw new ArgumentNullException(nameof(logger));
    }

    /// <summary>
    /// Create a new user account
    /// </summary>
    public async Task<CreateUserResponse> CreateUserAsync(CreateUserRequest request)
    {
      _logger.LogDebug("Calling UserManagement.createUser for email: {Email}", request?.Email);

      var response = await _messaging.CallAsync<CreateUserResponse>(
        ServiceTopic,
        "createUser",
        request
      );

      return response;
    }

    /// <summary>
    /// Retrieve a user by ID
```

```
        /// </summary>
        public async Task<User?> GetUserAsync(string userId)
        {
            _logger.LogDebug("Calling UserManagement.getUser for ID: {UserId}", userId);

            var response = await _messaging.CallAsync<User?>(
                ServiceTopic,
                "getUser",
                userId
            );

            return response;
        }

        /// <summary>
        /// List all users, optionally filtered by department
        /// </summary>
        public async Task<List<User>> ListUsersAsync(string? department = null)
        {
            _logger.LogDebug("Calling UserManagement.listUsers for department: {Department}",
                department ?? "(all)");

            var response = await _messaging.CallAsync<List<User>>(
                ServiceTopic,
                "listUsers",
                department
            );

            return response ?? new List<User>();
        }
    }
}
```

Step 4: Using the Generated C# Client

File: `Program.cs`

csharp

```csharp
using System;
using System.Threading.Tasks;
using Generated.UserManagement;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Platform.Messaging;

namespace CSharpUserClient
{
  class Program
  {
    static async Task Main(string[] args)
    {
      var host = Host.CreateDefaultBuilder(args)
        .ConfigureServices((context, services) =>
        {
            services.AddLogging();
            services.AddSingleton<IMessaging, MqttMessaging>();
            services.AddScoped<IUserManagementClient, UserManagementClient>();
        })
        .Build();

      using var scope = host.Services.CreateScope();
      var userClient = scope.ServiceProvider.GetRequiredService<IUserManagementClient>();
      var logger = scope.ServiceProvider.GetRequiredService<ILogger<Program>>();

      try
      {
         logger.LogInformation("Starting C# client demo...");

        // Create a new user
        var createRequest = new CreateUserRequest
        {
            Email = "john.doe@example.com",
            Name = "John Doe",
            Department = "Engineering"
        };

         logger.LogInformation("Creating user: {Email}", createRequest.Email);
        var createResponse = await userClient.CreateUserAsync(createRequest);

        if (createResponse.Success)
```

```
        {
            logger.LogInformation("User created successfully: {UserId}", createResponse.UserId);

            // Retrieve the created user
            var user = await userClient.GetUserAsync(createResponse.UserId);
            if (user != null)
            {
                logger.LogInformation("Retrieved user: {Name} ({Email}) in {Department}",
                    user.Name, user.Email, user.Department);
            }

            // List all engineering users
            var engineeringUsers = await userClient.ListUsersAsync("Engineering");
            logger.LogInformation("Found {Count} users in Engineering department",
                engineeringUsers.Count);

            foreach (var engUser in engineeringUsers)
            {
                logger.LogInformation("  - {Name} ({Email})", engUser.Name, engUser.Email);
            }
        }
        else
        {
            logger.LogError("Failed to create user: {Message}", createResponse.Message);
        }
    }
    catch (Exception ex)
    {
        logger.LogError(ex, "Error in user management demo");
    }
  }
 }
}
```

Step 5: Running the Cross-Language Integration

```bash
# User B runs their C# application
dotnet run

# Output:
# info: CSharpUserClient.Program[0]
#       Starting C# client demo...
# info: CSharpUserClient.Program[0]
#       Creating user: john.doe@example.com
# dbug: Generated.UserManagement.UserManagementClient[0]
#       Calling UserManagement.createUser for email: john.doe@example.com
# info: CSharpUserClient.Program[0]
#       User created successfully: 123e4567-e89b-12d3-a456-426614174000
# dbug: Generated.UserManagement.UserManagementClient[0]
#       Calling UserManagement.getUser for ID: 123e4567-e89b-12d3-a456-426614174000
# info: CSharpUserClient.Program[0]
#       Retrieved user: John Doe (john.doe@example.com) in Engineering
# dbug: Generated.UserManagement.UserManagementClient[0]
#       Calling UserManagement.listUsers for department: Engineering
# info: CSharpUserClient.Program[0]
#       Found 1 users in Engineering department
# info: CSharpUserClient.Program[0]
#         - John Doe (john.doe@example.com)
```

Key Benefits Demonstrated

1. Language Agnostic Development

- User A writes in Python with natural Python idioms
- User B writes in C# with native C# types and patterns
- No need to understand the implementation language of consumed services

2. Automatic Schema Generation

- Python type annotations automatically generate OpenRPC schema
- No manual schema writing or maintenance required
- Schema stays in sync with implementation changes

3. Type-Safe Client Generation

- C# client code is fully typed with validation attributes
- IntelliSense and compile-time checking work perfectly
- Runtime errors caught at development time

4. Zero API Maintenance

- No REST API endpoints to define or maintain
- No HTTP routing or serialization concerns
- Direct function-to-function communication

5. Seamless Integration

- C# client calls Python service transparently
- Automatic message serialization/deserialization
- Error handling and retry logic built-in

6. Development Workflow Efficiency

- User A deploys Python service with `platform-cli dev start`
- User B generates client with `platform-cli generate client`
- Both developers work independently with immediate integration

---

Schema Evolution Example

User A Updates Python Service

python

```python
# User A adds a new field to CreateUserRequest
@dataclass
class CreateUserRequest:
    email: str
    name: str
    department: Optional[str] = None
    phone_number: Optional[str] = None # NEW FIELD
    employee_id: Optional[str] = None # NEW FIELD
```

Automatic Schema Update

bash

```
# User A's rapid restart detects the change
# 📝 File changed: src/services/user_service.py
# 🔄 Regenerating OpenRPC schema from type annotations...
# 📋 Updated schema: schemas/UserManagement.openrpc.json
# ✅ Schema backward compatibility: MAINTAINED
# 🚀 UserManagement service deployed
# ✅ Rapid restart complete (3.8s)
```

User B Updates Client

bash

```
# User B regenerates client to get new fields
platform-cli generate client \
 --service UserManagement \
 --language csharp \
 --output ./Generated \
 --update

# Output:
# 📋 Loading updated schema: UserManagement.openrpc.json
# 🔍 Schema changes detected:
#   + CreateUserRequest.phone_number (optional)
#   + CreateUserRequest.employee_id (optional)
# ✅ Backward compatibility: MAINTAINED
# 🔄 Updating C# client code...
# ✅ C# client update complete
```

User B's existing code continues to work, but now has access to new optional fields:

csharp

```csharp
// Existing code still works
var createRequest = new CreateUserRequest
{
    Email = "jane.doe@example.com",
    Name = "Jane Doe",
    Department = "Marketing"
};

// New optional fields available
var enhancedRequest = new CreateUserRequest
{
    Email = "jane.doe@example.com",
    Name = "Jane Doe",
    Department = "Marketing",
    PhoneNumber = "+1-555-0123",  // New field
    EmployeeId = "EMP-001"       // New field
};
```

This example demonstrates how the platform enables true polyglot development where teams can use their preferred languages while maintaining type safety and automatic integration.