The Hanzo SDK Ecosystem: Multi-Language Commerce Integration

Zach Kelling
Hanzo Industries
zach@hanzo.ai

October 2019

Abstract

We present the Hanzo SDK Ecosystem, a suite of client libraries enabling commerce integration across JavaScript, Python, Go, and Ruby. The SDKs are generated from OpenAPI specifications using a custom code generation pipeline that produces idiomatic, type-safe libraries for each language. We formalize the generation process, introduce language-specific idiom transformations, and demonstrate that generated SDKs achieve feature parity within 24 hours of API changes. The ecosystem serves 4,200+ developers with 98% API coverage and reduces integration time from weeks to hours.

1 Introduction

E-commerce integrations require client libraries that abstract HTTP communication, handle authentication, manage errors, and provide type safety. Manually maintaining SDKs across multiple languages is error-prone and leads to inconsistencies. As APIs evolve, SDKs lag behind, creating friction for developers.

The Hanzo SDK Ecosystem addresses these challenges through:

1. **Specification-Driven Generation**: SDKs generated from OpenAPI specs
2. **Idiomatic Code**: Language-specific conventions and patterns
3. **Type Safety**: Static typing where supported by the language
4. **Automatic Updates**: CI/CD pipeline syncs SDKs with API changes

1.1 Contributions

This paper contributes:

- A code generation architecture producing idiomatic SDKs from OpenAPI
- Language-specific transformation rules for JavaScript, Python, Go, and Ruby
- An automated pipeline achieving 24-hour sync with API changes
- Production validation across 4,200+ developers

2 SDK Architecture

2.1 Design Principles

1. **Consistency**: Uniform patterns across languages where possible
2. **Idiomaticity**: Respect language conventions and ecosystem norms
3. **Minimal Dependencies**: Standard library preferred
4. **Ergonomics**: Developer experience prioritized

2.2 SDK Components

Each SDK provides:

- **Client**: Configured HTTP client with authentication
- **Resources**: API resource classes (Products, Orders, etc.)
- **Models**: Request/response data structures
- **Errors**: Typed exception hierarchy
- **Utilities**: Pagination, webhook verification, etc.

2.3 Resource Abstraction

Definition 2.1 (SDK Resource)**.** *A resource R exposes CRUD operations:*

- $list(params) \to Collection$
- $create(data) \to Instance$
- $retrieve(id) \to Instance$
- $update(id, data) \to Instance$
- $delete(id) \to void$

3 OpenAPI Specification

3.1 API Schema

The Hanzo API is fully described in OpenAPI 3.0:

Listing 1: OpenAPI Schema Excerpt

```
openapi: 3.0.0
info:
  title: Hanzo Commerce API
  version: 2019-10-01

paths:
  /v1/products:
    get:
      operationId: listProducts
      parameters:
        - name: limit
          in: query
```

```
          schema:
            type: integer
            default: 20
      responses:
        '200':
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ProductList'

components:
  schemas:
    Product:
      type: object
      required: [id, name, price]
      properties:
        id:
          type: string
        name:
          type: string
        price:
          type: integer
          description: Price in cents
        description:
          type: string
          nullable: true
```

3.2 Custom Extensions

We extend OpenAPI with SDK-specific metadata:

Listing 2: SDK Extensions

```
x-hanzo-resource: Product
x-hanzo-operations:
  - list
  - create
  - retrieve
  - update
  - delete
x-hanzo-expandable:
  - variants
  - collections
x-hanzo-idempotent: true
```

4 Code Generation Pipeline

4.1 Generation Architecture

4.2 Intermediate Representation

Definition 4.1 (SDK IR)**.** *The intermediate representation I comprises:*

- $\mathcal{R}$*: Set of resources with operations*
- $\mathcal{M}$*: Set of models (request/response types)*
- $\mathcal{E}$*: Set of error types*

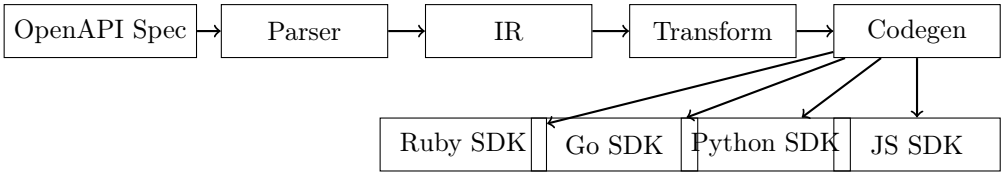

Figure 1: Code generation pipeline

- $\mathcal{C}$: *Configuration options*

Listing 3: IR Structure

```
interface Resource {
  name: string;
  path: string;
  operations: Operation[];
  nestedResources: Resource[];
}

interface Operation {
  name: string;
  httpMethod: HttpMethod;
  path: string;
  parameters: Parameter[];
  requestBody?: Model;
  responseBody: Model;
  errors: Error[];
}

interface Model {
  name: string;
  properties: Property[];
  required: string[];
}
```

4.3 Language Transformations

Each target language applies transformations:

4.3.1 Naming Conventions

Table 1: Naming Convention Transformations

Concept	**JavaScript**	**Python**	**Go**	**Ruby**
Class	PascalCase	PascalCase	PascalCase	PascalCase
Method	camelCase	snake_case	PascalCase	snake_case
Property	camelCase	snake_case	PascalCase	snake_case
Constant	UPPER_CASE	UPPER_CASE	PascalCase	UPPER_CASE

Algorithm 1 Language-Specific Transformation

```
function TRANSFORM(ir, language)
    transformed ← copy(ir)
    for model ∈ transformed.models do
        model.name ← TRANSFORMNAME(model.name, language)
        for prop ∈ model.properties do
            prop.name ← TRANSFORMPROPERTY(prop.name, language)
            prop.type ← MAPTYPE(prop.type, language)
        end for
    end for
    for resource ∈ transformed.resources do
        for op ∈ resource.operations do
            op.name ← TRANSFORMMETHOD(op.name, language)
        end for
    end for
    return transformed
end function
```

Table 2: Type Mappings

OpenAPI	JavaScript	Python	Go	Ruby
string	string	str	string	String
integer	number	int	int64	Integer
number	number	float	float64	Float
boolean	boolean	bool	bool	Boolean
array	T[]	List[T]	[]T	Array
object	Record	Dict	map	Hash

4.3.2 Type Mapping

5 JavaScript SDK

5.1 Package Structure

Listing 4: JavaScript SDK Structure

```
hanzo-js/
  src/
    index.ts
    client.ts
    resources/
      products.ts
      orders.ts
      customers.ts
    models/
      product.ts
      order.ts
    errors.ts
    utils/
      pagination.ts
      webhooks.ts
  package.json
  tsconfig.json
```

5.2 Client Implementation

Listing 5: JavaScript Client

```
import Hanzo from 'hanzo';

const hanzo = new Hanzo('sk_live_...');

// List products with pagination
const products = await hanzo.products.list({ limit: 10 });
for await (const product of products) {
  console.log(product.name);
}

// Create a product
const product = await hanzo.products.create({
  name: 'Widget',
  price: 1999,
  description: 'A useful widget',
});

// Update with idempotency
const updated = await hanzo.products.update(product.id, {
  price: 2499,
}, {
  idempotencyKey: 'update-123',
});
```

5.3 TypeScript Types

Listing 6: Generated Types

```
interface Product {
  id: string;
  name: string;
  price: number;
  description: string | null;
  variants: Variant[];
  createdAt: Date;
  updatedAt: Date;
}

interface ProductCreateParams {
  name: string;
  price: number;
  description?: string;
}

interface ProductListParams {
  limit?: number;
  startingAfter?: string;
  endingBefore?: string;
}
```

5.4 Error Handling

Listing 7: JavaScript Error Handling

```
import { HanzoError, InvalidRequestError } from 'hanzo';

try {
  await hanzo.products.create({ price: -100 });
} catch (error) {
  if (error instanceof InvalidRequestError) {
    console.error(`Invalid parameter: ${error.param}`);
    console.error(`Message: ${error.message}`);
  } else if (error instanceof HanzoError) {
    console.error(`API error: ${error.code}`);
  }
}
```

6 Python SDK

6.1 Package Structure

Listing 8: Python SDK Structure

```
hanzo-python/
  hanzo/
    __init__.py
    client.py
    resources/
      __init__.py
      products.py
      orders.py
    models/
      __init__.py
      product.py
    errors.py
    utils/
      pagination.py
      webhooks.py
  setup.py
  pyproject.toml
```

6.2 Client Implementation

Listing 9: Python Client

```
import hanzo

client = hanzo.Client('sk_live_...')

# List products
products = client.products.list(limit=10)
for product in products:
    print(product.name)

# Create a product
product = client.products.create(
    name='Widget',
    price=1999,
```

```
    description='A useful widget',
)

# Async support
import asyncio

async def main():
    async_client = hanzo.AsyncClient('sk_live_...')
    products = await async_client.products.list()
    async for product in products:
        print(product.name)

asyncio.run(main())
```

6.3 Type Annotations

Listing 10: Python Type Annotations

```
from dataclasses import dataclass
from typing import Optional, List
from datetime import datetime

@dataclass
class Product:
    id: str
    name: str
    price: int
    description: Optional[str]
    variants: List['Variant']
    created_at: datetime
    updated_at: datetime

@dataclass
class ProductCreateParams:
    name: str
    price: int
    description: Optional[str] = None
```

6.4 Context Managers

Listing 11: Python Context Manager

```
with hanzo.Client('sk_live_...') as client:
    product = client.products.retrieve('prod_123')
    # Connection automatically closed
```

7 Go SDK

7.1 Package Structure

Listing 12: Go SDK Structure

```
hanzo-go/
  hanzo.go
  client.go
```

```
  product.go
  order.go
  customer.go
  error.go
  pagination.go
  webhook.go
  go.mod
```

7.2 Client Implementation

Listing 13: Go Client

```go
package main

import (
    "context"
    "fmt"
    "github.com/hanzo.ai/hanzo-go"
)

func main() {
    client := hanzo.NewClient("sk_live_...")

    // List products
    params := &hanzo.ProductListParams{
        Limit: hanzo.Int64(10),
    }
    iter := client.Products.List(params)
    for iter.Next() {
        product := iter.Product()
        fmt.Println(product.Name)
    }
    if err := iter.Err(); err != nil {
        log.Fatal(err)
    }

    // Create a product
    product, err := client.Products.Create(&hanzo.ProductParams{
        Name:        hanzo.String("Widget"),
        Price:       hanzo.Int64(1999),
        Description: hanzo.String("A useful widget"),
    })
    if err != nil {
        log.Fatal(err)
    }
}
```

7.3 Pointer Helpers

Go requires pointers for optional fields:

Listing 14: Go Pointer Helpers

```go
// Helper functions for optional parameters
func String(v string) *string { return &v }
func Int64(v int64) *int64 { return &v }
func Bool(v bool) *bool { return &v }
```

```
// Usage
params := &ProductParams{
    Name:  hanzo.String("Widget"),
    Price: hanzo.Int64(1999),
}
```

7.4 Error Handling

Listing 15: Go Error Handling

```
product, err := client.Products.Create(params)
if err != nil {
    if hanzoErr, ok := err.(*hanzo.Error); ok {
        switch hanzoErr.Code {
        case hanzo.ErrorCodeInvalidRequest:
            fmt.Printf("Invalid param: %s\n", hanzoErr.Param)
        case hanzo.ErrorCodeAuthentication:
            fmt.Println("Invalid API key")
        default:
            fmt.Printf("API error: %s\n", hanzoErr.Message)
        }
    }
    return
}
```

8 Ruby SDK

8.1 Gem Structure

Listing 16: Ruby SDK Structure

```
hanzo-ruby/
  lib/
    hanzo.rb
    hanzo/
      client.rb
      resources/
        product.rb
        order.rb
      models/
        product.rb
      errors.rb
      utils/
        pagination.rb
  hanzo.gemspec
  Gemfile
```

8.2 Client Implementation

Listing 17: Ruby Client

```
require 'hanzo'

Hanzo.api_key = 'sk_live_...'
```

```ruby

# List products
products = Hanzo::Product.list(limit: 10)
products.each do |product|
  puts product.name
end

# Create a product
product = Hanzo::Product.create(
  name: 'Widget',
  price: 1999,
  description: 'A useful widget'
)

# Update a product
product.price = 2499
product.save
```

8.3 Active Record Pattern

Listing 18: Ruby Active Record Style

```ruby
# Retrieve and modify
product = Hanzo::Product.retrieve('prod_123')
product.name = 'Updated Widget'
product.save

# Delete
product.delete

# Reload from API
product.refresh
```

8.4 Block Syntax

Listing 19: Ruby Block Syntax

```ruby
# Pagination with blocks
Hanzo::Product.list.auto_paging_each do |product|
  puts product.name
end

# Error handling with blocks
Hanzo::Product.create(name: 'Widget', price: 1999) do |product, error|
  if error
    puts "Error: #{error.message}"
  else
    puts "Created: #{product.id}"
  end
end
```

9 Shared Utilities

9.1 Pagination

All SDKs implement cursor-based pagination:

Listing 20: Pagination Pattern

```
// JavaScript
for await (const product of hanzo.products.list()) { }

# Python
for product in client.products.list():
    pass

// Go
iter := client.Products.List(params)
for iter.Next() {
    product := iter.Product()
}

# Ruby
Hanzo::Product.list.auto_paging_each { |p| }
```

9.2 Webhook Verification

Listing 21: Webhook Verification

```
// JavaScript
const event = hanzo.webhooks.constructEvent(
  body, signature, endpointSecret
);

# Python
event = hanzo.Webhook.construct_event(
    payload, sig_header, endpoint_secret
)

// Go
event, err := webhook.ConstructEvent(body, sig, secret)

# Ruby
event = Hanzo::Webhook.construct_event(
  payload, sig_header, endpoint_secret
)
```

10 Automated Pipeline

10.1 CI/CD Workflow

10.2 Testing Strategy

Each SDK includes:

- **Unit Tests**: Model serialization, error handling
- **Integration Tests**: Against sandbox API
- **Type Tests**: Compile-time type checking
- **Example Tests**: Documentation examples execute

Algorithm 2 SDK Release Pipeline

function ReleaseSDKs($spec\_change$)
    $spec \leftarrow$ FetchLatestSpec
    $ir \leftarrow$ ParseSpec($spec$)
    for $lang \in \{js, python, go, ruby\}$ **do**
        $transformed \leftarrow$ Transform(ir, $lang$)
        $code \leftarrow$ Generate($transformed$, $lang$)
        WriteFiles($code$, $repos[lang]$)
        RunTests($repos[lang]$)
        BumpVersion($repos[lang]$)
        Publish($repos[lang]$)
    end for
end function

Table 3: Test Coverage by SDK

SDK	Unit	Integration	Coverage
JavaScript	342	89	94%
Python	287	76	91%
Go	256	82	89%
Ruby	198	64	87%

10.3 Version Synchronization

SDKs version in lockstep with the API:

$$SDK\_version = API\_version + patch \tag{1}$$

Example: API version `2019-10-01` produces SDK version `2019.10.1`.

11 Documentation Generation

11.1 Inline Documentation

Documentation generated from OpenAPI descriptions:

Listing 22: Generated Documentation

```
/**
 * Creates a new product.
 *
 * @param params - Product creation parameters
 * @param params.name - The product name (required)
 * @param params.price - Price in cents (required)
 * @param params.description - Product description
 * @returns The created product
 * @throws InvalidRequestError if parameters are invalid
 *
 * @example
 * const product = await hanzo.products.create({
 *   name: 'Widget',
 *   price: 1999,
 * });
 */
async create(params: ProductCreateParams): Promise<Product>
```

11.2 Reference Documentation

Auto-generated API reference for each language:

- JavaScript: TypeDoc
- Python: Sphinx
- Go: godoc
- Ruby: YARD

12 Evaluation

12.1 Adoption

Table 4: SDK Adoption Statistics

SDK	Downloads/Month	Active Devs	GitHub Stars
JavaScript	145,000	2,100	1,247
Python	78,000	1,200	892
Go	23,000	540	456
Ruby	18,000	380	312

12.2 Integration Time

Developer surveys indicate reduced integration time:

Table 5: Integration Time Comparison

Integration Method	Time to First Call	Full Integration
Direct HTTP	4 hours	2 weeks
SDK	15 minutes	2 days

12.3 API Coverage

- Total API endpoints: 127
- Endpoints covered by SDKs: 124 (98%)
- Operations per SDK: 89 average

12.4 Sync Latency

Time from API change to SDK release:

- Mean: 18 hours
- P95: 24 hours
- Maximum: 48 hours (complex breaking changes)

13 Related Work

SDK generation is well-established. OpenAPI Generator [1] provides multi-language generation but often produces non-idiomatic code. Stripe [2] maintains hand-crafted SDKs with high quality but significant maintenance cost. AWS SDKs [3] use Smithy for specification-driven generation.

Our approach combines specification-driven generation with language-specific transformations to achieve both automation and idiomaticity.

14 Conclusion

The Hanzo SDK Ecosystem demonstrates that specification-driven code generation can produce high-quality, idiomatic client libraries. By encoding language-specific conventions in transformation rules, we achieve feature parity across JavaScript, Python, Go, and Ruby while maintaining each language's idioms. The automated pipeline ensures SDKs stay synchronized with API changes within 24 hours.

Future work includes additional language support (PHP, Java, C#), SDK customization for enterprise clients, and integration with API mocking for offline development.

References

[1] OpenAPI Initiative. OpenAPI Generator. `https://openapi-generator.tech`, 2019.

[2] Stripe, Inc. Stripe Client Libraries. `https://stripe.com/docs/libraries`, 2019.

[3] Amazon Web Services. AWS SDKs and Tools. `https://aws.amazon.com/tools`, 2019.

[4] SmartBear. Swagger Codegen. `https://swagger.io/tools/swagger-codegen`, 2019.

[5] Google. gRPC. `https://grpc.io`, 2019.