MODAPTO [101091996]: Modular Manufacturing and Distributed Control via Interoperable Digital Twins

11.3.2 Implementing orchestration logic and error handling

2025-08-11

Implementing orchestration logic and error handling in the MODAPTO Orchestrator involves translating workflow designs into executable code and configurations that coordinate service interactions while gracefully handling exceptions and failures. This process leverages the Orchestrator's user interface and JavaScript capabilities to create robust, reliable orchestrations.

Implementing Basic Orchestration Logic

To implement orchestration logic in the MODAPTO Orchestrator:

1. **Create Microservice Structure**:
   - Access the Orchestrator UI and click "Create New" to create a new microservice
   - Define the microservice name, description, and visibility (public or private)
   - Add operations to the microservice by expanding the operations section
2. **Configure Individual Operations**:
   - Define operation IDs, names, and descriptions
   - Specify whether operations should start automatically
   - Designate one operation as the default entry point if appropriate
   - Select the appropriate connector for each operation
3. **Configure Connector-Specific Settings**:
   - Complete the initialization configuration for each connector
   - Define the execution configuration, including endpoints, methods, and parameters
   - Configure input parameters that will be exposed to users of the orchestration

Implementing Advanced Orchestration with JavaScript

For complex orchestrations requiring conditional logic, data transformations, or service coordination:

1. **Create a JavaScript Engine Operation**:
   - Add an operation with the JavaScript Engine Connector
   - Define input parameters needed by the orchestration logic
   - Configure the JavaScript execution environment
2. **Implement Orchestration Script**:
   - Use the JavaScript language to define orchestration logic
   - Access input parameters through the standardized 'input' object
   - Call other services using the callMicroservice function:

javascript

Copy

```
var result = callMicroservice("microserviceId", "operationId", {param1: "value1"});
```

2. Implement conditional logic based on service results:

javascript

Copy

```
if (result.status === “success”) {
  // Call next service in the happy path
} else {
  // Implement error handling or alternative path
}
```

2. Transform data between service calls as needed:

javascript

Copy

```
var transformedData = {
  newParam: result.data.originalParam + ” (processed)”
};
```

2. Return final results using the out function:

javascript

Copy

```
out({result: “Orchestration completed”, data: finalResult});
```

Implementing Error Handling

Effective error handling is critical for robust orchestrations:

1. **Error Detection**:
   - Implement validation checks on service inputs and outputs
   - Detect timeouts and connectivity failures
   - Identify business logic errors based on service responses
2. **Error Recovery Strategies**:
   - Retry Logic: Attempt failed operations multiple times with appropriate backoff

javascript

Copy

```
function retryOperation(maxAttempts) {
  var attempts = 0;
  var result;
  while (attempts < maxAttempts) {
    try {
      result = callMicroservice("serviceId", "operationId", params);
      if (result.success) return result;
    } catch (e) {
      // Log error
    }
    attempts++;
    // Implement exponential backoff
    sleep(Math.pow(2, attempts) * 1000);
  }
  return {success: false, error: "Max attempts reached"};
}
```

2. Alternative Paths: Define alternative services or workflows when preferred paths fail
3. Compensation Logic: Undo previous operations when a workflow cannot complete
4. Graceful Degradation: Provide partial results when full processing is impossible
5. **Error Reporting**:
    - Provide clear error messages with context information
    - Include details about which service failed and why
    - Offer suggestions for resolution where applicable

Testing and Debugging Orchestrations

The MODAPTO Orchestrator provides tools for testing and debugging orchestrations:

1. **Testing Interface**:

- Use the “Test a Call” button to access the testing interface
- Provide sample inputs for the orchestration
- Review the raw JSON output to verify correct behavior
- Use the testing function to validate specific operations

2. **Debugging Techniques**:
   - Add logging statements to JavaScript code to track execution flow
   - Test individual operations before combining them into workflows
   - Use the Call Status Check Algorithm to validate operation results
   - Monitor operation status indicators (green/yellow/red) in the UI

Once tested and debugged, orchestrations can be started using the “Start” button, making them available for execution through their REST endpoints. For complex orchestrations, operations can be configured to start automatically when the system initializes.

References

1. MODAPTO Consortium. (2023). Orchestrator User Manual. MODAPTO Project Documentation.
2. Alonso, G., Casati, F., Kuno, H., & Machiraju, V. (2004). Web Services: Concepts, Architectures and Applications. Springer.
3. Nygard, M. T. (2007). Release It!: Design and Deploy Production-Ready Software. Pragmatic Bookshelf.
4. Mozilla Developer Network. (2023). JavaScript Guide. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide