Error Handling

Source File: generated-documents\technical-design\error-handling.md
Generated: 08/07/2025 at 09:44:04
Generated by: Requirements Gathering Agent - PDF Converter

ErrorHandling

Generated by adpa-enterprise-framework-automation v3.1.6
Category: technical-design
Generated: 2025-07-05T17:07:20.964Z
Description:

Self-Charging Electric Vehicle (SCEV) Project: Error Handling Guidelines

These guidelines define the error handling strategy for the SCEV project, encompassing all software and hardware components. The goal is to create a robust and reliable system that gracefully handles errors, minimizes downtime, and provides informative feedback to both users and developers.

1. Error Handling Strategy:

Our strategy employs a layered approach combining preventative measures, robust error detection, and comprehensive handling at various levels:

- **Preventative:** Rigorous code reviews, static analysis, unit/integration testing, and comprehensive simulations will be used to identify and mitigate potential errors before deployment. Design for failure will be a core principle.
- **Detection:** Each component will include robust error detection mechanisms, using assertions, exception handling (try-catch blocks), and sensor monitoring for hardware components.
- **Handling:** Errors will be handled gracefully, prioritizing data integrity, system stability, and user experience. This includes logging, reporting, and appropriate recovery actions. The severity of the error will determine the response (e.g., retry, fallback, fail-safe mode).
- **Reporting:** Comprehensive error reporting will allow developers to quickly identify and fix issues. This includes detailed logs, alerts, and dashboards.
- **User Experience:** User-facing error messages will be clear, concise, and helpful, guiding users on how to resolve the issue or contact support.

2. Error Categories:

Errors will be categorized for better analysis and prioritization:

- **Hardware Errors:** Failures in solar panels, regenerative suspension, TEG units, battery, or other physical components. These will be further categorized by component and type of failure (e.g., sensor malfunction, short circuit, overheating).
- **Software Errors:** Bugs in the AI-powered EMU, communication protocols, data processing, or other software components. These will be categorized by module and type (e.g., runtime exception, logic error, data corruption).
- **Communication Errors:** Failures in communication between different components (e.g., EMU and sensors, EMU and vehicle control system).
- **Environmental Errors:** Extreme weather conditions (e.g., intense heat, heavy snow) affecting energy generation.

- **User Errors:** Incorrect user input or actions leading to errors (e.g., incorrect settings, attempted unauthorized access).

3. Error Logging:

- **Centralized Logging:** All errors will be logged to a central logging system, using a consistent format (e.g., JSON). This system should be highly available and scalable. Consider using a distributed logging solution like ELK stack or similar.
- **Log Levels:** Log messages will use standardized log levels (e.g., DEBUG, INFO, WARNING, ERROR, CRITICAL) to indicate severity.
- **Contextual Information:** Logs will include detailed contextual information such as timestamps, component IDs, error codes, stack traces (for software errors), and sensor readings (for hardware errors).
- **Log Rotation:** Logs will be rotated regularly to prevent disk space exhaustion.
- **Security:** Log data will be secured appropriately to prevent unauthorized access.

4. Error Reporting:

- **Automated Alerts:** Critical errors will trigger automated alerts via email, SMS, or other notification systems to relevant personnel.
- **Dashboards:** A dashboard will provide real-time monitoring of error rates, allowing for proactive identification of potential issues.
- **Error Tracking System:** An error tracking system (e.g., Sentry, Rollbar) will be used to track and manage reported errors, facilitating debugging and resolution.

5. Recovery Procedures:

- **Fail-Safe Modes:** Components should have fail-safe modes to prevent catastrophic failures. For example, if a sensor fails, the system might default to a conservative estimate.
- **Redundancy:** Critical components might have redundant backups to ensure continued operation in case of failure.

- **Automatic Recovery:** The system should attempt automatic recovery whenever possible (e.g., restarting a failed component, retrying a failed operation).
- **Manual Intervention:** For errors that cannot be automatically resolved, clear instructions for manual intervention will be provided.

6. Retry Mechanisms:

- **Exponential Backoff:** Retrying failed operations with exponentially increasing delays will help avoid overwhelming the system during transient failures.
- **Maximum Retries:** A maximum number of retries should be defined to prevent infinite retry loops.
- **Jitter:** Adding random jitter to retry delays can help avoid synchronized retries that could exacerbate the problem.

7. Circuit Breakers:

- **Implement Circuit Breakers:** Circuit breakers will prevent repeated calls to failing services, protecting the system from cascading failures. This is particularly important for external dependencies.

8. User Error Messages:

- **Clear and Concise:** User error messages should be clear, concise, and easy to understand.
- **Actionable:** Messages should provide users with actionable steps to resolve the issue.
- **Helpful:** Messages should provide helpful context and guidance.
- **Localization:** Support for multiple languages.

9. Monitoring and Alerts:

- **Real-time Monitoring:** Real-time monitoring of key system metrics (e.g., error rates, resource utilization, energy generation) will be implemented.
- **Threshold-based Alerts:** Alerts will be triggered when key metrics exceed predefined thresholds.

- **Automated Escalation:** Automated escalation procedures will be defined for critical errors.

10. Troubleshooting Guide:

A comprehensive troubleshooting guide will be created and maintained, providing step-by-step instructions for resolving common errors. This guide will be accessible to both users and support personnel.

Specific Considerations for SCEV:

- **Battery Health:** Implement robust monitoring and error handling for battery health, including overcharging, overheating, and deep discharge protection.
- **Energy Harvesting:** Implement error handling for variations in energy generation due to weather conditions or road surfaces.
- **AI-Powered EMU:** Implement robust error handling and logging for the machine learning models within the EMU, including model retraining and fallback mechanisms.

These guidelines will be reviewed and updated regularly as the project progresses and new challenges emerge. Adherence to these guidelines is crucial for the success of the SCEV project.
