Luxoft University Connect Program - Phase 2

Project Title : **AI-Based ARXML Semantic Comparator Project**

Institution : GM University

Luxoft Sponsor : Pradeep M K

Luxoft SPOC : Seenevasaraj S

Table of Contents

1. Project Summary

AUTOSAR (AUTomotive Open System ARchitecture) is a worldwide standard in the automotive industry. OEMs and suppliers rely on ARXML files to describe ECU software components, communication interfaces, and system configurations. However, comparing ARXML files manually is tedious and error-prone, especially as files can grow to gigabytes in size.

The AI-Based ARXML Semantic Comparator project addresses this by providing a web-based application that enables engineers to perform semantic comparisons of ARXML files. Unlike simple diff tools, this comparator ignores reordering and focuses on meaningful changes in signals, ports, and attributes. The system integrates AI models such as Gemini (primary) and LLaMA 3 (fallback) to generate natural language explanations of differences. The output is delivered in multiple formats: Tree View for structural overview, Tabular View for detailed analysis, and Text Summary for quick insights.

This project directly contributes to faster review cycles, improved traceability, and higher reliability in AUTOSAR-based development workflows.This reduces manual effort by 60–70% compared to line-by-line diff

2. Software Requirements Derivation

2.1 Functional Requirements

Req ID	Requirement	Description	Status
FR_001	File Upload & Parsing	Allow upload of `.arxml` and `.xsd` files; use streaming to parse large files.	Specified
FR_002	Semantic Comparison	Compare ARXML files ignoring reordering; focus on UUIDs and attributes.	Specified
FR_003	AI Explanations	Generate natural language summary of differences using Gemini/LLaMA.	Specified

FR_004	Multi-Format Output	Present results in Tree View, Table View, and Text Summary.	Specified
FR_005	Chatbot (Optional)	RAG-based chatbot for ARXML queries.	Optional
FR_06	Export Results to CSV/Excel	Allow users to download the tabular comparison results in a machine-readable format for further analysis	Specified

2.2 Non-Functional Requirements

Req ID	Requirement	Description
NFR_001	Performance	The system must handle multi-GB files without crashing or freezing.
NFR_002	Scalability	The system must support multiple users performing comparisons simultaneously.
NFR_003	Reliability	The system must ensure async background processing with status monitoring.
NFR_004	Usability	The system must provide an intuitive UI with progress indicators and clear error messages.
NFR_005	Maintainability	The codebase must be modular, well-documented, and include a README and API documentation.
NFR_006	Compliance	The project must follow the Luxoft Way of Working, Agile methodology, and use JIRA for logging.

3. Architecture

The system follows a modular client-server architecture with AI integration:

High-Level Architecture:

- Frontend: React + TailwindCSS for modern, responsive UI; D3.js for visualization.
- Backend: Django REST framework for API endpoints; Celery + Redis for async processing.
- AI Layer: Gemini 1.5 Pro (cloud API), LLaMA 3 (fallback via Ollama).
- Storage: Disk-based handling of uploads with chunked streaming.

Low-Level Workflow:

File Upload & Initial Storage:

- **User Action:** User uploads `File A.arxml`, `File B.arxml`, and an optional `schema.xsd` through the web interface.
- **Frontend Action:** Frontend sends files as a multipart form data request to the Django backend.
- **Backend Action:** The Django backend receives the files and saves them to a designated local disk location (e.g., `/media/uploads`). It creates a new `ComparisonJob` record in the SQL database, storing metadata (file paths, user ID, timestamp) and setting the status to 'UPLOADED'. The backend then sends a lightweight, instant response back to the frontend.

User Selects Comparison Type:

- **Frontend Action:** The frontend presents the user with three explicit options on the UI: "Full Comparison," "Schema Validation Only," and "Semantic Comparison Only."
- **User Action:** The user selects their desired comparison type and confirms.
- **Backend Action:** The frontend sends the selected comparison type to a specific backend API endpoint. The backend updates the `ComparisonJob` record with this choice and changes the status to 'QUEUED'.

Asynchronous Task Enqueueing:

- **Backend Action:** Based on the user's selection, the Django backend enqueues a corresponding task (e.g., `run_full_comparison_task`,

`run_schema_validation_task`) with `Celery`, passing the unique job ID. This task is placed in the `Redis` message queue.
- **Frontend Action:** The frontend polls the backend for the job's status, providing a "Processing..." indicator to the user.

Worker Processing (Celery Worker):

- **Worker Action:** A `Celery` worker picks up the task from the queue.
- **Conditional Step 1 (Schema Validation):** If the user selected "Full Comparison" or "Schema Validation Only," the worker streams and validates the `.arxml` files against the `.xsd` schema using `xmlschema`.
    - **Outcome:** If validation fails, the job status is updated to 'FAILED_VALIDATION', and an error message is generated. If it passes, the process continues.
- **Conditional Step 2 (Semantic Diff):** If the user selected "Full Comparison" or "Semantic Comparison Only," the worker proceeds to the core comparison.
    - **Parser:** The worker uses `lxml.etree.iterparse()` to stream and build tree structures for each ARXML file, ensuring memory efficiency.
    - **Diff Engine:** The custom Python-based diff engine compares the two tree structures semantically, ignoring node order and matching components based on unique identifiers like UUIDs and attributes. The output is a structured JSON object representing the differences.

AI-Powered Explanation:

- **Worker Action:** The worker sends the structured JSON diff to the `Gemini 1.5 Pro` API (or a local `LLaMA 3` instance for fallback). The request includes a carefully crafted prompt to guide the AI.
- **AI Service Action:** The AI model processes the prompt and generates a natural language text summary of the changes.

Storing Results and Display Preparation:

- **Worker Action:** The worker saves the structured JSON diff and the AI-generated text summary back into the `ComparisonJob` record in the PostgreSQL database. It updates the status to 'COMPLETED'.
- **Frontend Action:** The frontend's polling detects the 'COMPLETED' status and triggers a request to fetch the results from the backend.

Final Output Display:

- **Frontend Action:** The frontend receives the results and uses `React-Tree` or `D3.js` to render the interactive Tree View, a data table component for the Tabular View, and a text component for the Text Summary.

4. Coding Guidelines

The codebase adheres to Python PEP8 standards and Luxoft coding practices:

- Functions have docstrings specifying purpose, inputs, and outputs.
- Global constants in UPPER_CASE.
- Error handling with try-except for file parsing and API calls.
- Logging framework integrated for debugging and monitoring.
- GitHub used for version control; commits must be descriptive.

Example Function Header:

- Function Name : parse_arxml
- DESCRIPTION : Streams and parses ARXML files into tree structure
- PURPOSE : Enable efficient semantic comparison
- INPUT : file_path (str)
- OUTPUT : Parsed tree object
- AUTHOR : Team Luxoft Connect Phase 2

5. Test Case Design

Example Test Case:

Test ID: TC_001
Requirement Reference: FR_001
Description: Verify ARXML file upload and parsing.
Preconditions: Valid .arxml file available.
Procedure:
1. Navigate to upload page.
2. Select valid .arxml file.
3. Submit file.
Expected Result: File is uploaded, parsed, and confirmed in backend.

Test ID: TC_002

Requirement Reference: FR_002
Description: Verify semantic diff detects changes in UUID.
Preconditions: Two ARXML files with differing UUID.
Expected Result: Diff engine identifies change and displays in all formats.

6. Traceability Matrix

SW Requirement	Test Case
FR_001 File Upload & Parsing	TC_001
FR_002 Semantic Comparison	TC_002, TC_003
FR_003 AI Explanations	TC_004
FR_004 Multi-Format Output	TC_005
FR_005 Chatbot	TC_006 (optional)

7. Milestone Plan

T0: Statement of Work shared by Luxoft.

Week 1-2 : Kick-off meeting, project role allocation, initial setup.

Week 3-4 : **Frontend development** – UI wireframes, React, initial Tree/Table/Text placeholders.

Week 5-6 : Backend development – file upload and parser implementation.

Week 7-8 : Async task management (Celery + Redis).

Week 9-10 : AI integration (Gemini + fallback LLaMA).

Week 11-12 : Testing phase – unit tests, integration tests, bug fixes.

Week 13-15: Documentation finalization, final demo to Luxoft.

Review Points:
- Weekly mentor connects.
- Monthly sponsor review.
- Final demo and project closure at Luxoft.

8. Conclusion

The AI-Based ARXML Semantic Comparator is a scalable, reliable, and AI-enhanced solution for the automotive industry. By combining efficient semantic diff algorithms with natural language explanations, it reduces manual review time, improves clarity, and ensures quality in AUTOSAR-based workflows. With Luxoft's Way of Working guiding the process, the project meets professional standards in software development lifecycle, documentation, and delivery.