UNIVERSITI MALAYA

WIF3005
ASSIGNMENT 2

IMPACT ANALYSIS: HOME ASSISTANT
[ENTITY REGISTRY MODULE]

DR. NASUHA BINTI MOHD DAUD

NAME	STUDENT ID
AHMAD SHAZWY DANIEL BIN AHMAD SAZELI	U2001270

1.0 Addressed Component

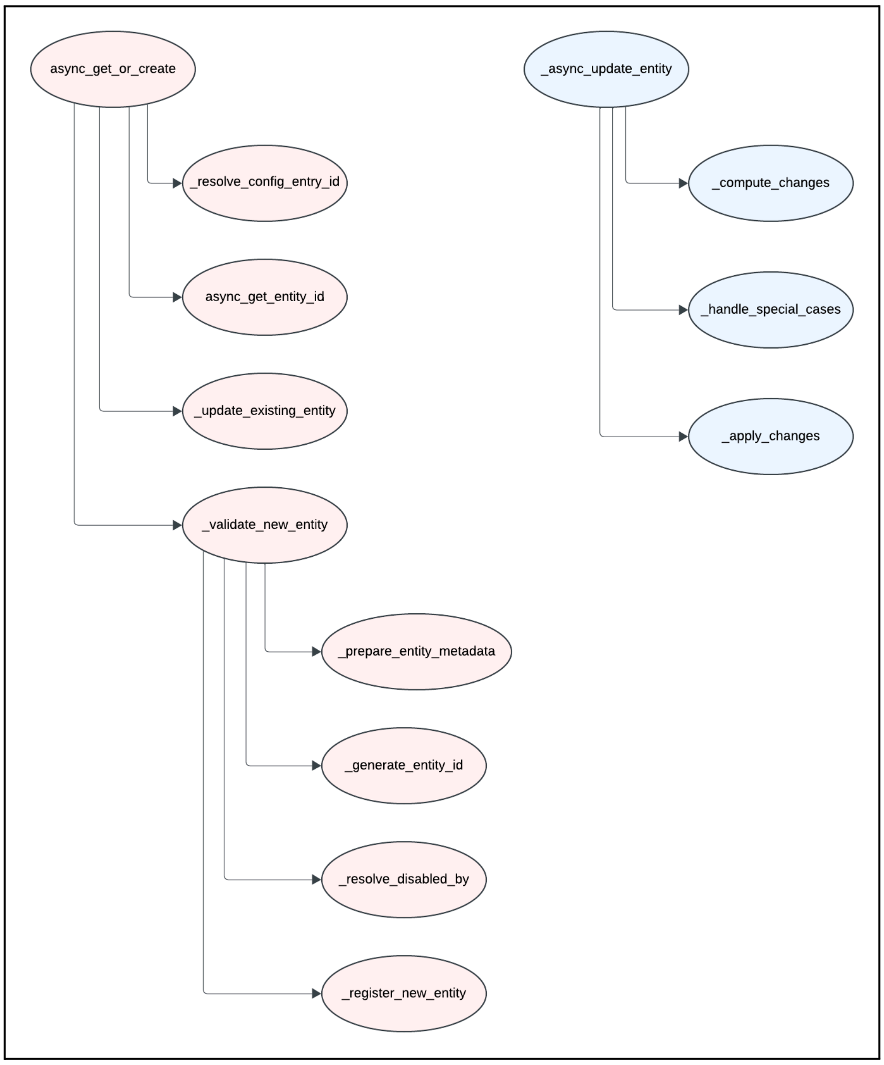
The component addressed in this analysis is the **Entity Registry** module, specifically the methods async_get_or_create and _async_update_entity within the file helpers/entity_registry.py.

These methods are central to the functionality of the entity registry, responsible for managing and maintaining entities in the Home Assistant system.

Their complexity and nested structures needed refactoring to enhance maintainability, readability, and ease of debugging.

2.0 Call Graph

For this analysis, a **Call Graph** was created to illustrate the interactions between the targeted methods and other functions in the system.



Graph 2.0: Entity Registry Call Graph

The graph captures the following details:

- **Direct and Indirect Calls**:
    - It highlights the functions called by async_get_or_create and _async_update_entity, as well as the functions that indirectly call these methods.
    - Direct calls, such as _resolve_config_entry_id and _compute_changes, are explicitly invoked within the targeted methods.
    - While indirect calls, such as _prepare_entity_metadata are invoked through other functions, providing support for extended functionality.
- **Completeness**:
    - The graph comprehensively maps all known function dependencies within the Entity Registry module and their connections to the system's broader functionality.

3.0 Insights From Impact Analysis

The Call Graph provided critical insights into the **Entity Registry's** functional structure:

- **Identified High-Impact Areas**: Functions heavily dependent on async_get_or_create and _async_update_entity were flagged for thorough testing post-refactoring to ensure functionality is preserved.
- **Simplified Refactoring Plan**: The graph revealed opportunities to modularize nested logic into reusable helper functions without disrupting dependent components.
- **Reduced Risk**: By visualizing the function interactions, we could pinpoint areas with minimal dependencies, allowing safer and faster refactoring iterations.
- **Testing Scope**: The graph helped define the testing scope by identifying all functions and modules directly or indirectly affected by the targeted methods.

This impact analysis graph ultimately guided a structured approach to refactoring while minimizing risks and ensuring system integrity.