Energy Modeling System Enhancements: Parameter Validation & Agent Handover

Executive Summary

We identified and fixed several critical issues in the energy modeling multi-agent system, particularly around parameter validation, user input prioritization, and inter-agent communication. The system now properly collects all required parameters, respects user input over extracted values, and maintains clean context handovers between agents. The complete workflow from model creation to analysis and reporting now functions correctly.

Table of Contents

Initial Issue Assessment

We identified several key issues in the initial system:

1. **Location Inference Bug**: Nova defaulted to "Bosnia" even when other locations were specified
2. **Incomplete Parameter Collection**: Parameter collection stopped after the first parameter
3. **Parameter Priority Conflict**: System prioritized extracted parameters over user-provided ones
4. **Function Mapping Issues**: "analyze_results" function was not properly mapped
5. **Parameter Signature Errors**: Functions rejected parameters they received from other agents

The issues affected the workflow in several ways:

```
User Prompt: "build a model for Spain"
↓
Nova extracts location: "Spain", generation: "solar" (default)
↓
Emil ignores location or replaces with "Bosnia" (incorrect behavior)
↓
Model created with wrong location
```

Parameter Collection Enhancements

Problem

The parameter collection only requested one parameter at a time and didn't continue to collect all required parameters.

```
Prompt: "build a model"
System: "Please provide location"
User: "UK"
System: "Please specify a generation type" (error displayed, but no input collected)
```

Solution

Implemented a parameter collection loop in `emil.py` that continues until all required parameters are collected:

```
# In handle_task_async in emil.py
while not validation["success"] and validation.get("missing"):
    print(f"Emil needs additional information for {task.name}")

    # Handle missing parameters interactively
    from .simplified_parameter_collection import get_missing_parameters_simple_async

    # Collect missing parameters
    collected_params = await get_missing_parameters_simple_async(
        function_name=task.function_name,
        missing_params=validation["missing"],
        initial_args=task.args
    )

    # Update task arguments
    for k, v in collected_params.items():
        task.args[k] = v

    # Re-verify with the new parameters
    validation = await self.verify_parameters_async(task.function_name, task.args)
```

This ensures all parameters are collected before proceeding with model creation.

Agent Handover Improvements

Problem

Context handovers between agents contained irrelevant information from previous tasks, leading to potential confusion and parameter conflicts.

Solution

1. Implemented clean context handovers that only include relevant information for the current task
2. Modified the `verify_parameters_async` function to properly validate required parameters:

```
@log_function_call
async def verify_parameters_async(self, function_name: str, task_args: dict) -> dict:
    # Special case for basic Emil requests with just a prompt
    if function_name == 'process_emil_request' and task_args.get('prompt'):
        # For energy modeling, check if location is missing or "Unknown"
        location = task_args.get('location')
        if not location or location == "Unknown":
            return {
                "success": False,
                "missing": ["location"],
                "message": "Please specify a location (country or region) for the energy model
            }

        # Check if generation type is missing
        generation = task_args.get('generation') or task_args.get('generation_type')
        if not generation:
            return {
                "success": False,
                "missing": ["generation"],
                "message": "Please specify a generation type (solar, wind, hydro, etc.)."
            }

        return {"success": True, "missing": [], "message": "Prompt and essential parameters pr
```

Function Parameter Fixes

Problem

The `analyze_results` function was failing with:

```
Error executing analyze_results: analyze_results() got an unexpected keyword argument 'full_pr
```

Solution

Modified the `analyze_results` function signature to accept and handle the `full_prompt` parameter:

```
@log_function_call
def analyze_results(kb, prompt=None, full_prompt=None, analysis_type="basic", model_file=None,
    """
    Analyzes energy model results.

    Parameters:
        kb (KnowledgeBase): Knowledge base
        prompt (str, optional): The original prompt
        full_prompt (str, optional): The full original prompt (added parameter)
        analysis_type (str): Type of analysis to perform
        model_file (str): Path to the model file to analyze
        model_details (dict): Details about the model

    Returns:
        dict: Analysis results
    """
    # Function implementation...
```

Testing and Results

We developed and executed test cases to verify our fixes:

Test Case 1: Complete Parameter Collection Loop

Prompt: `build a model`

Expected & Actual Results:

- System prompted for location: "all" ✅
- System prompted for generation type: "all" ✅
- Model created with correct parameters ✅
- File created: "All_all_Electricity_20250526_131916.xml" ✅

Test Case 2: User Input Priority

Prompt: `build a model`

Expected & Actual Results:

- System prompted for location: "france" ✅
- System prompted for generation type: "bio" ✅
- User input was respected over extracted values ✅

- File created: "France_bio_Electricity_20250526_132015.xml" ✅

Test Case 3: Multi-Agent Workflow

Prompt: `build a hydro model for Spain, analyze the results, write a report`

Expected & Actual Results:

- System created hydro model for Spain ✅
- Analysis was performed correctly ✅
- Analysis completed with 4 findings ✅
- Report generated with correct parameters ✅

Code Implementation Details

File Mappings

The changes were implemented across several files:

1. **src/agents/emil.py**
   - Modified `verify_parameters_async` to properly validate parameters
   - Enhanced `handle_task_async` with parameter collection loop
2. **src/core/functions_registery.py**
   - Fixed `process_emil_request` to prioritize user input
   - Updated `analyze_results` to accept additional parameters
3. **src/agents/nova.py**
   - Improved `create_task_list_from_prompt_async` to better handle analysis tasks

Parameter Flow Example

```
User Input: "build a model"
↓
Nova creates task with empty parameters
↓
Emil verifies parameters, finds missing "location"
↓
System prompts: "I need the 'location' for this task"
User provides: "france"
↓
Emil verifies again, finds missing "generation"
↓
System prompts: "I need the 'generation' for this task"
```

```
User provides: "bio"
↓
Emil verifies parameters again, all required parameters present
↓
Emil extracts parameters from prompt (locations=Unknown, generation_types=solar)
↓
Emil prioritizes user input (location=france, generation=bio)
↓
Emil creates model with user-provided parameters
↓
Result: France_bio_Electricity_*.xml created
```

Parameter Priority Fix

The most critical fix was in the `process_emil_request` function, ensuring that user-provided parameters always take precedence:

```python
# For generation type - FIXED to always prioritize user input
if passed_generation:
    # Always use user-provided generation parameter
    final_generation = passed_generation
elif extracted_params['generation_types']:
    final_generation = extracted_params['generation_types'][0]
else:
    # Never default to a value, require explicit parameter
    return {
        "status": "error",
        "message": "Generation type is required"
    }
```

This ensures that user input is always respected, addressing a critical usability issue.

Conclusion

The enhancements implemented in this session have significantly improved the robustness and usability of the energy modeling system. The system now:

1. Correctly collects all required parameters
2. Properly respects and prioritizes user input
3. Maintains clean context handovers between agents
4. Successfully completes the full workflow from model creation to analysis and reporting

These improvements ensure that the system is more reliable, user-friendly, and produces accurate results that truly reflect the user's intent.