A Modular Translation Layer for Static and Deterministic Payload Translation Between Heterogeneous Imperative-Based Agents Using LLMs

11/07/2025

Leon Cherchye

The Need for Payload Translation Between Agents

- Common in IoT, smart cities, industrial automation, emergency services, etc.
- Agents are often **heterogeneous**: different data formats, protocols, vocabularies.
- Without payload translation, agents **cannot understand** each other → collaboration breaks down.

Syntactic Discrepancy vs. Semantic Ambiguity

- Syntactic Discrepancy: different encoding/structure (e.g. JSON vs. XML, or two JSON schemas).
- Semantic Ambiguity: different representation for the same concept (e.g. “temp_Celsius” & “ambient_temp_degC”) or same representation for different concepts
- Standards help—but often lose detail

Syntactic Discrepancy

```
{
    metadata: {
        Type: INFORM,
    },
    body: {
        Room: Living,
        Temperature: 20,
        Heater_on: True,
        Extra: {
            Timer: 50
        }
    }
}
```

```
{
    metadata: {
    },
    body: {
        Type: INFORM,
        Room: Living,
        Status: {
            Temperature: 20,
            Heater_on: True,
            Timer: 50
        }
    }
}
```

Semantic Ambiguity

```
{
    metadata: {
        Type: INFORM,
    },
    body: {
        Room: Living,
        Temperature: 20,
        Heater_on: True,
        Extra: {
            Timer: 50
        }
    }
}
```

```
{
    metadata: {
    },
    body: {
        Tipo: INFORMAR,
        Sala: Sala de Estar,
        Estado: {
            Temperatura: 20,
            Aquecedor_ligado: True,
            Temporizador: 50
        }
    }
}
```

Semantic Ambiguity: Data Types

```
{
    metadata: {
        Type: INFORM,
    },
    body: {
        Room: Living,
        Temperature: 20,
        Heater_on: True,
        Extra: {
            Timer: 50
        }
    }
}
```

```
{
    metadata: {
    },
    body: {
        Tipo: INFORMAR,
        Sala: Sala de Estar,
        Estado: {
            Temperatura: 20.00,
            Aquecedor_ligado: 1,
            Temporizador: 50.00
        }
    }
}
```

Goal and Our Solution

- **Main Research Question**: How can a modular translation layer enable accurate and scalable payload translation between heterogeneous imperative-based agents without requiring modifications to their internal logic?
- **Design Goals**:
  - A **modular translation layer** deployed alongside agents.
  - No changes required to agents' original behavior code.
  - **Deterministic**, **efficient**, **distributed**, and **scalable**.
  - No information loss in translation

Intelligent Agents & MAS – Core Concepts

- Autonomous
- Reactive
- Proactive
- Perceives and acts in an environment.
- Agent Oriented Programming (AOP)
- Multiple agents in a shared environment.
- Agents may cooperate or compete.
- Benefits: Scalability, flexibility, fault tolerance, task distribution.

Agent Oriented Programming Approaches

Approach	Description Type	Example Systems	Strong in...
Rule-based	IF-THEN rules	Rql, Drools, CLIPS, PyKnow	Reasoning, decision-making
Imperative	Functional logic	SPADE, JADE	Full control, scripting
FSM (Finite State Machine)	State transitions	JADEX, Transitions	Reactive systems
BDI (Belief-Desire-Intention)	Cognitive model	Jason, JADEX	Intention-based planning
Declarative/Logical	Logical inference	Prolog, GOLOG	Planning, semantics
Workflow-based	Process/step plans	Camunda, JADE flows	Business and service workflows
Ontology-based	Semantic knowledge	CoBrA, OntoMAS, Onto2JaCaMo	Interoperability, web agents

Agent Communication

- Agent Communication Languages (ACL)
  - Outer language > Inner language > ontology/vocabulary
- **Protocols** standarize Outer language and provide structure for Inner language and ontology use
  - FIPA ACL, XMPP, KQML
- **Payload** = Inner language + ontology
  - The agent specific part inside a message

Assumptions

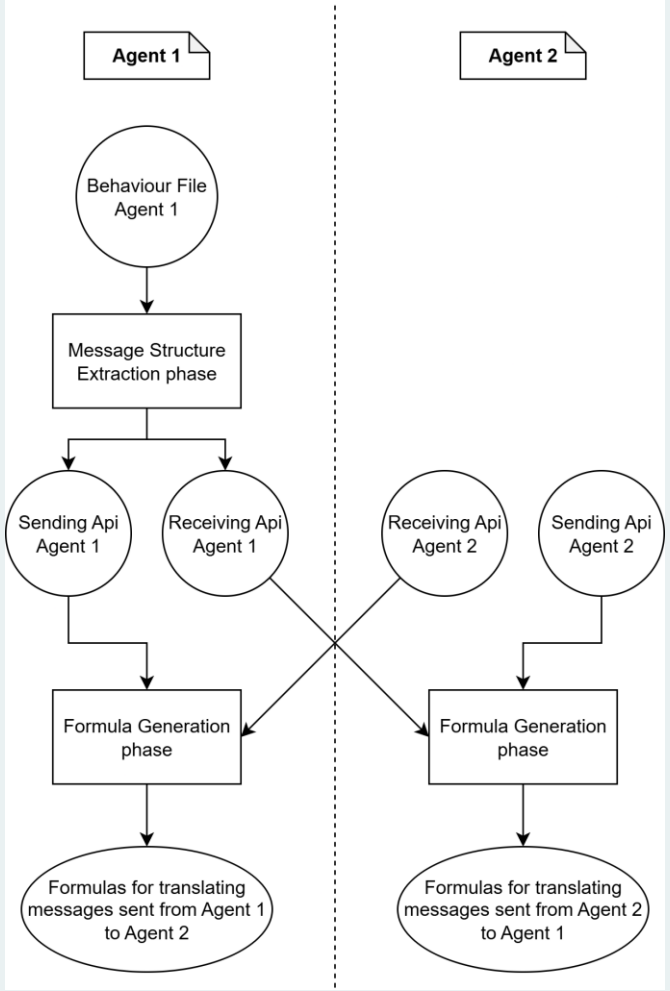
- Aligned Interaction/Expectiations Assumption
- No Optional Fields Assumption
- Access to LLM assumptions
    - Locally
    - External service

Design Approach

- Generating a static **Formula** for efficiency
  - No LLM during runtime
  - Generate formulas for all future translations
  - Template.json (structure) + key- and voc-mapping.csv's (semantics)

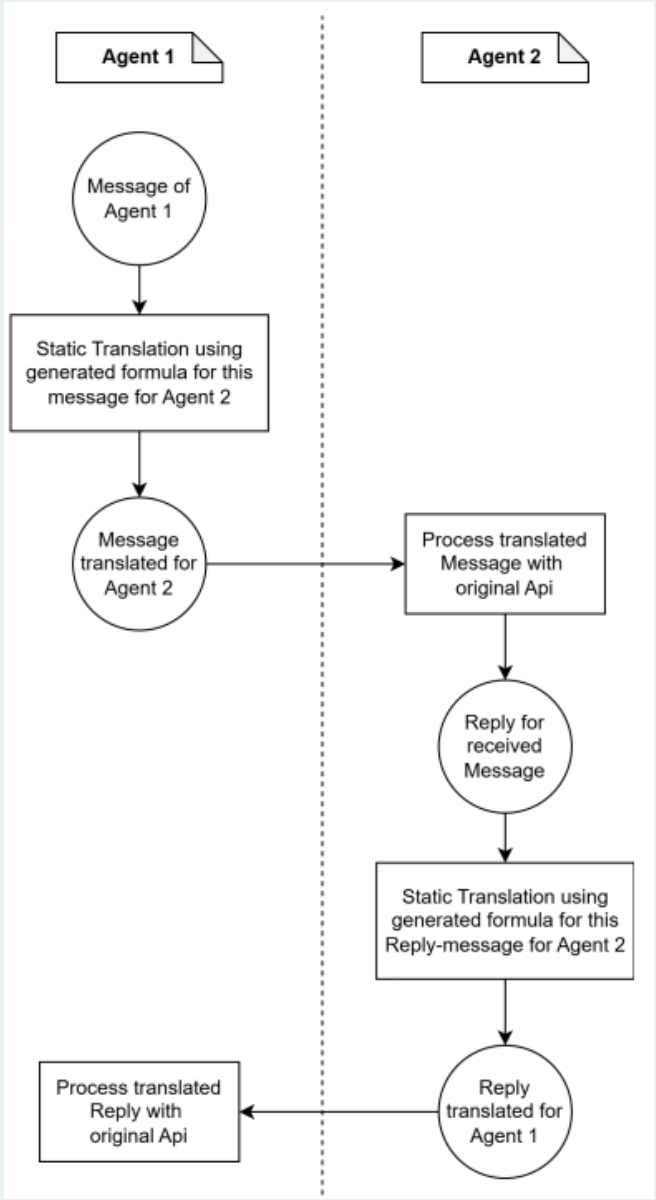
  → Deterministic translation
- Generate formula **locally** for distribution
  - Generate for each agent pair
  - No intermediate format → no information loss

Translation Workflow



← **Before** runtime translation

During runtime translation →



Message Structure Extraction Phase

- Goal: Extract all sending- and receiving messages of agent. Including message structure, keys, datatypes and vocabulary where necessary. Generate metadata for this information needed for generating the formulas.

1. LLM extracts all necessary information
2. Stability based filtering
3. LLM generates key explanations
    - No data types
4. LLM generates message explanations
5. Structure information in files for next phase

- Template.json for each message

Template.json File

```
{
    metadata: {
     },
     body: {
          Tipo: INFORMAR,
          Sala: Sala de Estar,
          Estado: {
              Temperatura: 20.00,
              Aquecedor_ligado: 1,
              Temporizador: 50.00
          }
    }
}
```

```
{
    metadata: {
     },
     body: {
          Tipo: INFORMAR,
          Sala: None,
          Estado: {
              Temperatura: None,
              Aquecedor_ligado: None,
              Temporizador: None
          }
    }
}
```

Formula Generation Phase

1. Message Mapping
    - Sending of agent to Receiving of another agent
    - Message intention allignment
    - One-to-Many mapping (keys and negative descriptions)
2. Key Mapping between mapped messages
    - Matched based on purpose, meaning, and role of the key
3. Vocabulary Mapping between mapped messages
    - Same mapping as in Key Mapping should be implemented

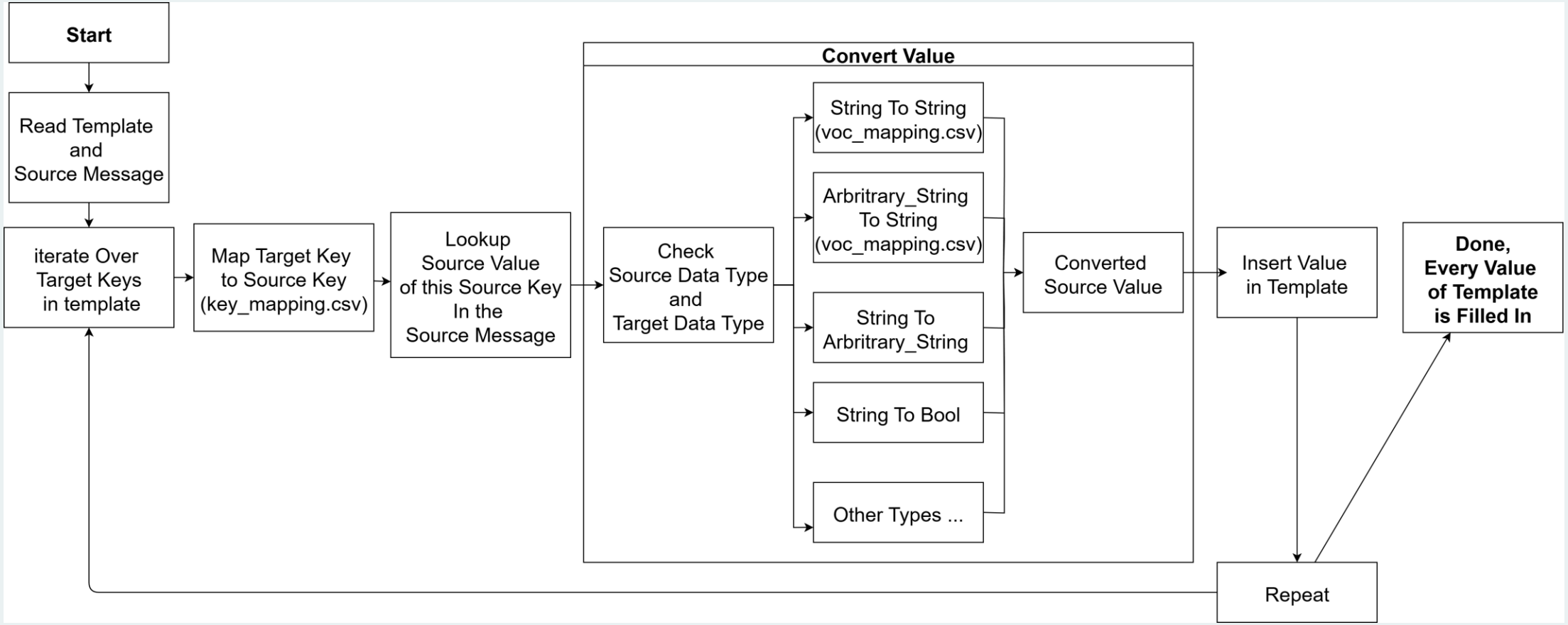
- Stability based filtering (all parts)
- Between each mapped messages: key- and voc_mapping.csv file

Runtime Translation Phase

1. Identify the receiving agent
2. Identify the message being send by the agent
3. Retrieve this uniquely indentified formula
4. Translate using the formula

- Supported data types for conversion: String, Arbritrary_string, Boolean, Integer and Float
- Integrate this functionality directly in send() function → No modifications needed in agent itself

Runtime Translation Phase



Example

Sala → Key_mapping.csv file → Room

Living → Voc_mapping.csv file → Sala de Estar

Type converter

```
{
    metadata: {
        Type: INFORM,
    },
    body: {
        Room: Living,
        Temperature: 20,
        Heater_on: True,
        Extra: {
            Timer: 50
        }
    }
}
```

```
{
    metadata: {
    },
    body: {
        Tipo: INFORMAR,
        Sala: Sala de Estar,
        Estado: {
            Temperatura: None,
            Aquecedor_ligado: None,
            Temporizador: None
        }
    }
}
```

Example

Temperatura → Key_mapping.csv file → Temperature

Voc_mapping.csv file

Int, 20 → Type converter → Float, 20.00

```
{
    metadata: {
        Type: INFORM,
    },
    body: {
        Room: Living,
        Temperature: 20,
        Heater_on: True,
        Extra: {
            Timer: 50
        }
    }
}
```

```
{
    metadata: {
    },
    body: {
        Tipo: INFORMAR,
        Sala:  Sala de Estar,
        Estado: {
            Temperatura:  20.00,
            Aquecedor_ligado: None,
            Temporizador: None
        }
    }
}
```

Test Cases

Test case 1

- Perfect string-to-string conversion
- Imperfect string-to-string conversion
- Arbitrary string filtering
- Redundant source messages
- Extracting from multiple files
- Structural differences
- Key name differences

Test case 2

- Boolean-to-integer conversion
- Integer-to-boolean conversion
- Integer-to-float conversion
- Float-to-integer conversion
- One-to-many message mapping
- Structural differences
- Key name differences

Test case 3

- Combination of both test cases

Result Test Cases (No Filtering)

- Gemini 2.5 Flash model
- 30 runs

Summary of Execution Time (s) and Agent Success Rate

	Test case 1		Test case 2		Test case 3		
Test Part \ Agent	**DeliveryAgent**	**PlannerAgent**	**ControllerAgent**	**HomeAgent**	**PlannerControllerAgent**	**DeliveryHomeAgent**	**Average**
Message Structure Extraction	14,38	11,22 **(28/30)**	14,31 **(27/30)**	12,06 **(29/30)**	20,00 **(26/30)**	20,61 **(27/30)**	15,43
Message Mapping	14,16	9,61	9,23	12,74	11,34	22,87 **(29/30)**	13,33
Key Mapping	13,32	10,96	10,54	11,24	8,96	21,13	12,69
Vocabulary Matching	8,55	N/A	N/A	N/A	N/A	8,84	8,70
Full process	50,41	31,79 **(28/30)**	34,08 **(27/30)**	36,04 **(29/30)**	40,30 **(26/30)**	64,61 **(26/30)**	42,87
Formula Generation Phase	36,03	20,57	19,77	23,98	20,30	52,84 **(29/30)**	**28,92**

This table shows the execution times for each agent, measured in seconds. Each row represents how much time is spent on each process step. Additional rows show the total execution time for the full process and the time spent on the formula generation phase. Green cells indicate a success rate of 30 out of 30. Red cells indicate failure, with the actual success rate shown in brackets.

Result Test Cases (Filtering Where Necessary)

- Gemini 2.5 Flash model
- 30 runs
- Filter threshold = 2

Summary of Execution Time (s) and Agent Success Rate with Stability Filtering Applied (Threshold = 2)

	Test case 1		Test case 2		Test case 3		
Test Part \ Agent	**DeliveryAgent**	**PlannerAgent**	**ControllerAgent**	**HomeAgent**	**PlannerControllerAgent**	**DeliveryHomeAgent**	**Average**
Message Structure Extraction	14,38	22,66	32,27	23,44	38,51	45,31	29,43
Message Mapping	14,16	9,61	9,23	12,74	11,34	46,07	17,19
Key Mapping	13,32	10,96	10,54	11,24	8,96	21,13	12,69
Vocabulary Matching	8,55	N/A	N/A	N/A	N/A	8,84	8,70
Full process	50,41	43,23	52,04	47,42	58,81	112,51	60,74
Formula Generation Phase	36,03	20,57	19,77	23,98	20,30	76,04	**32,78**

This table shows the execution times for each agent, measured in seconds, Each row represents how much time is spent on each process step, Additional rows show the total execution time for the full process and the time spent on the formula generation phase, Green cells indicate a success rate of 30 out of 30, Dark green cells represent agents for which a stability-based filtering mechanism was applied with a threshold of 2, because they did not achieve 100% success in the previous table,

- Message: 7 to 8 mapping
- Key: 24 to 11 mapping
- Vocabulary: 17 to 17 mapping

- Accurate and scalable

Conclusion

With integrating LLMs for payload translation we achieved:

- Accurate translation across different message structures and vocabularies
- No changes to internal logic or code
- **Three-phase design** ensures flexibility:
    1. Message Structure Extraction
    2. Formula Generation
    3. Runtime Translation
- Runtime translation is fast and deterministic
- Design is scalable, distributed and prevents information loss due to the translation

Future Improvements

- Exploring the Effectiveness of Fine-Grained vs. Single Query Extraction in LLMs
- Output Validation of The Extracted Structure via Logs
- Optional keys
- Function-Aware Vocabulary Mapping
- Handling of Lists and Dictionaries in Translation
- Unit Conversion
- Error Propagation
- Dynamic Selection in One-to-Many Message Mapping
- Extension to XML-Compatible Content Languages
- Many-to-One Mappings
- Portability Across Programming Languages

Are there any questions?

Thank you for Listening