Multi-step Iterative Automated Domain Modeling with Large Language Models

6th Workshop on Artificial Intelligence and Model-driven Engineering
Co-located with MODELS. September 2024. Linz, Austria.

Authors: Yujing Yang[1], Boqi Chen[1], Kua Chen[1], Gunter Mussbacher[1], Dániel Varró[1,2]

Presenter: Boqi Chen

[1]Electrical and Computer Engineering, McGill University, Montreal, Canada
[2]Department of Computer and Information Science (IDA), Linköping University, Linköping, Sweden

Table of Contents

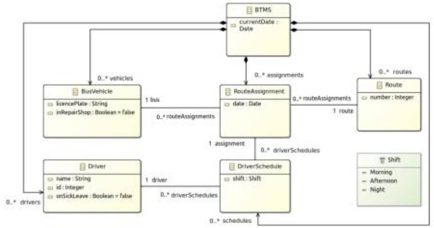
Challenge: Domain Model Creation

- **Performed manually by software engineers**
  - Time consuming
  - High variation
- **Existing (non-LLM) automated approaches**
  - Requires some level of human interaction
  - Domain elements are extracted mostly at sentence level

Natural Language Description

A city is using the Bus Transportation Management System (BTMS) to simplify the day-to-day activities related to the city's public bus system.
The BTMS keeps track of a driver's name and automatically assigns a unique ID to each driver. A bus route is identified by a unique number that is determined by city staff, while a bus is identified by its unique licence plate. The highest possible number for a bus route is 9999, while a licence plate number may be up to 10 characters long, inclusive. For up to a year in advance, city staff assigns buses to routes. Several buses may be assigned to a route per day. Each bus serves at the most one route per day but may be assigned to different routes on different days. Similarly, for up to a year in advance, city staff posts the schedule for its bus drivers. For each route, there is a morning shift, an afternoon shift, and a night shift. A driver is assigned by city staff to a shift for a particular bus on a particular day. The BTMS offers city staff great flexibility, i.e., there are no restrictions in terms of how many shifts a bus driver has per day. It is even possible to assign a bus driver to two shifts at the same time.
The current version of BTMS does not support the information of bus drivers or buses to be updated – only adding and deleting is supported. However, BTMS does support indicating whether a bus driver is on sick leave and whether a bus is in the repair shop. If that is the case, the driver cannot be scheduled or the bus cannot be assigned to a route. For a given day, an overview shows – for each route number – the licence plate number of each assigned bus, the entered shifts and the IDs and names of the assigned drivers. If a driver is currently sick or a bus is in the repair shop, the driver or bus, respectively, is highlighted in the overview.

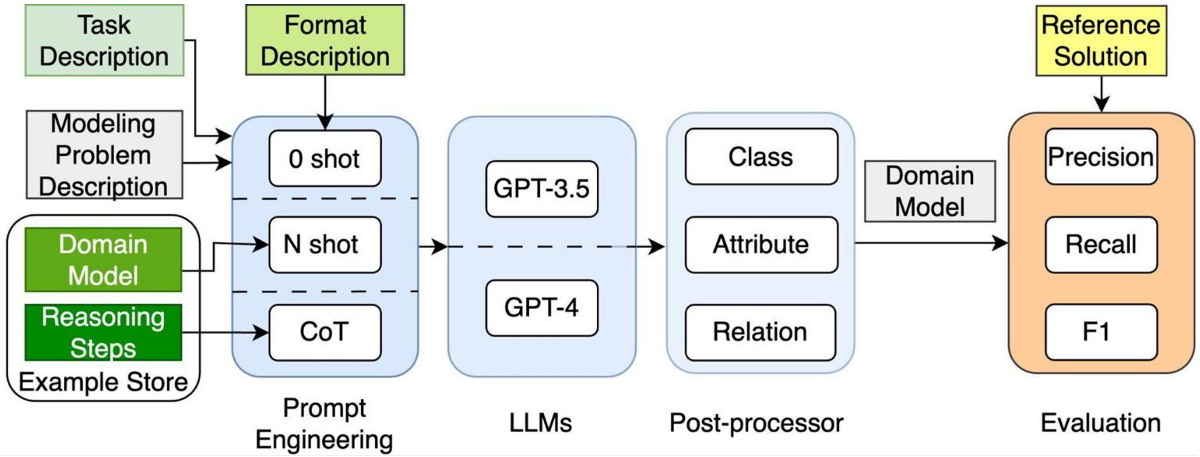
Domain Model



Background: Large Language Models (LLMs)

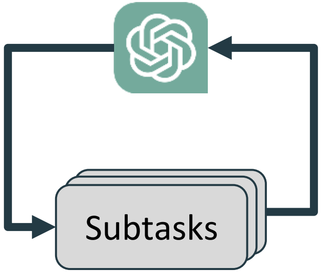
- LLMs are natural language processing methods designed for text generation.
- Basic mechanism: given a sequence, LLMs predict next token.
- Advantages of using LLM:
    - **few-shots learners**
    - Large and diverse

Previous Approach: Single Step Generation with LLMs [7]

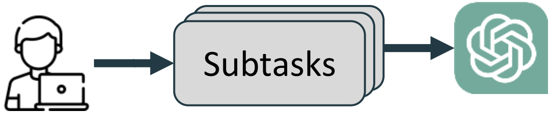


- *High precision* but *lower recall* for classes, attributes, and relationships
- No integration of *modeling practice*
- No modeling *patterns* identified, e.g., Player-role pattern
- **One-time effort with LLMs**

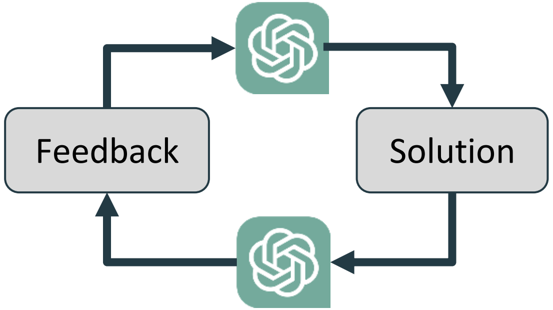
Practice from LLM Research



Iterative,
multi-step [19, 22]



Involving domain
knowledge [10]

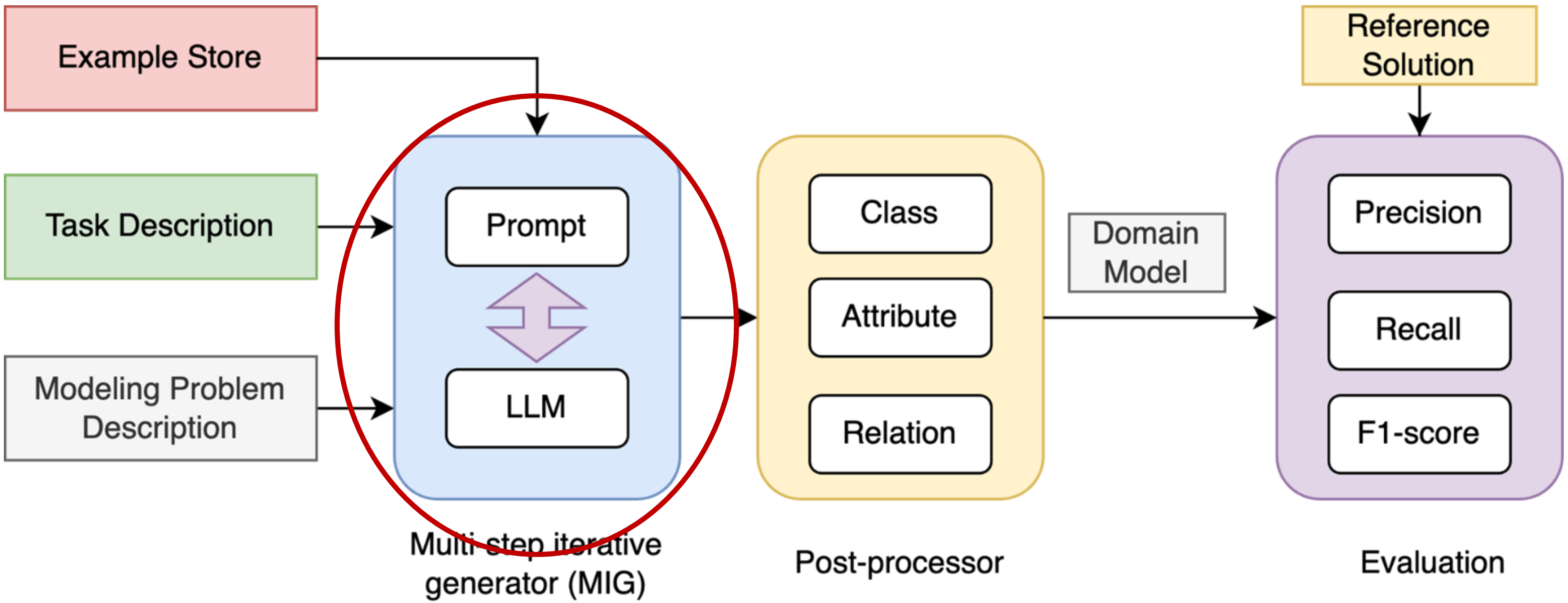


Self-feedback
[13,18]

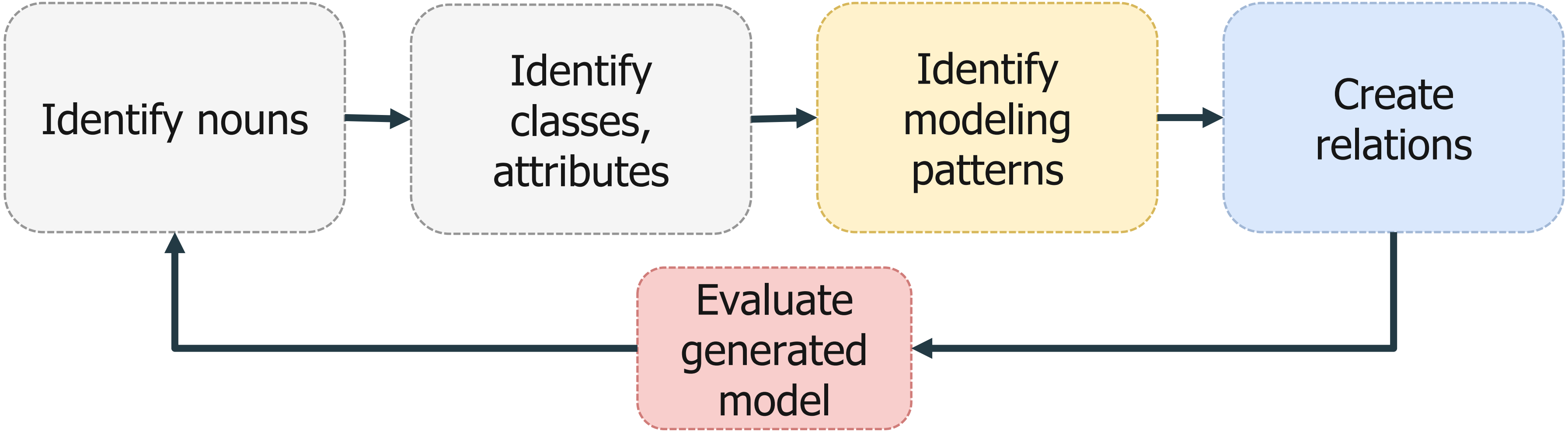
Reference numbers are from the paper

Table of Contents

Approach: Architecture



How Do Engineers Create Domain Models?

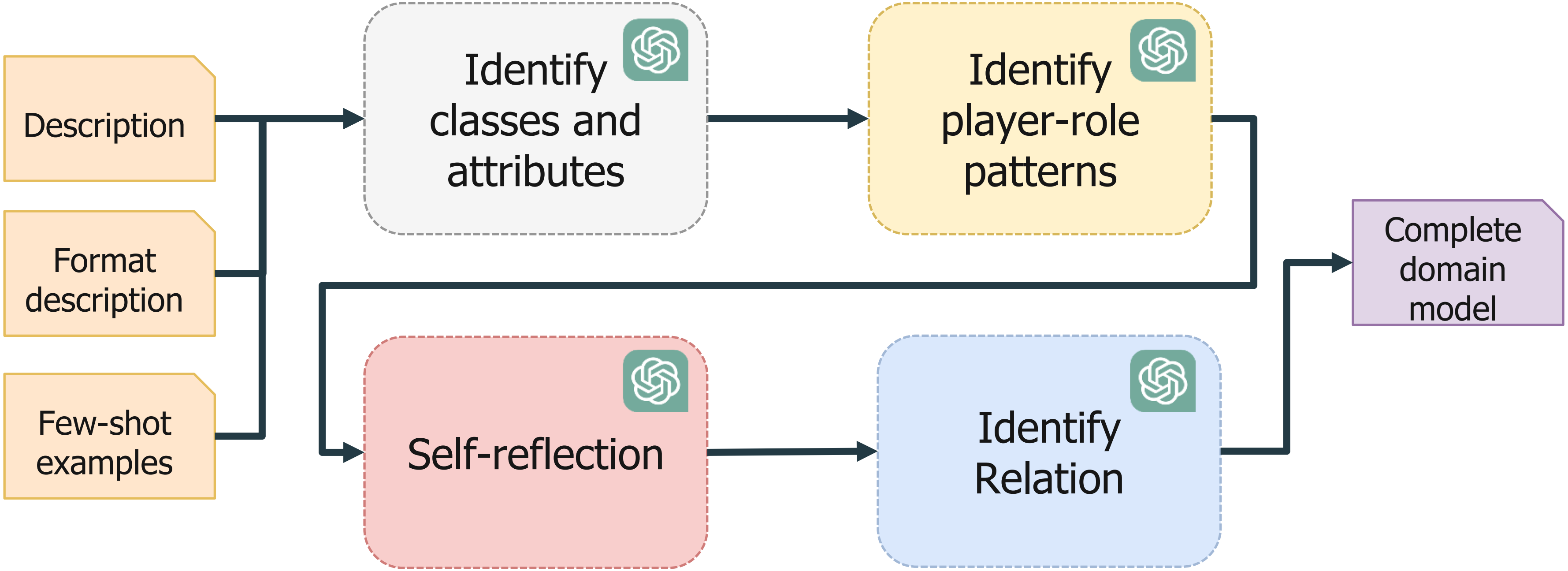


Enable LLMs to follow the same process!

Overview

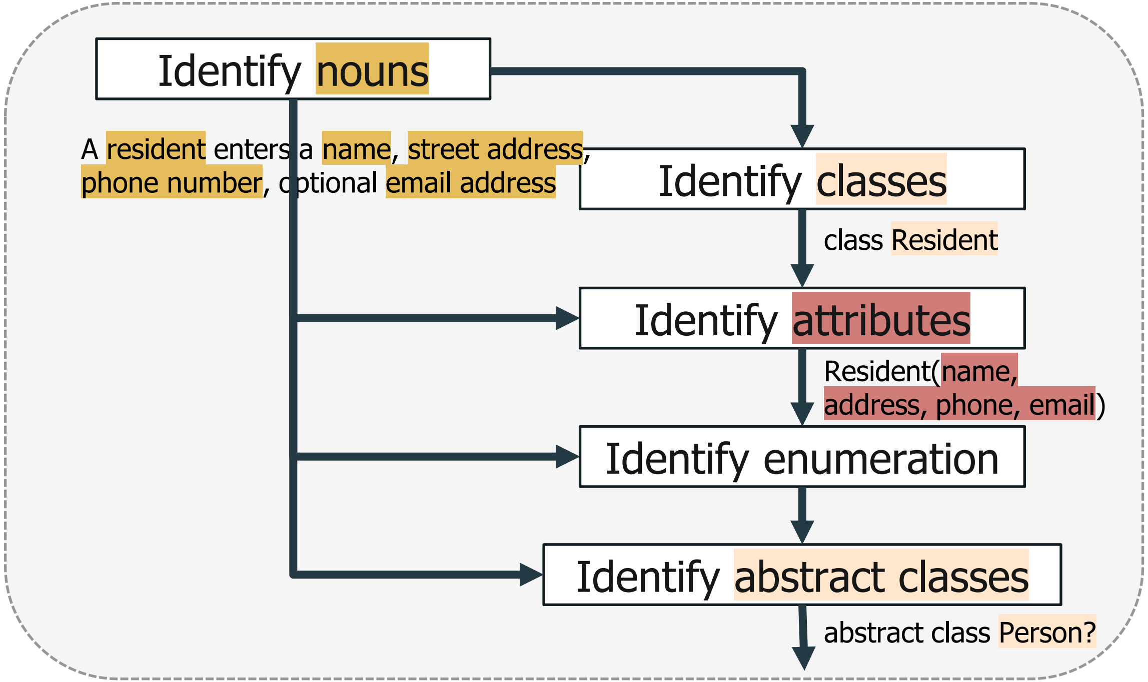
Identify classes and attributes → Identify player-role patterns → Self-reflection → Identify Relation

Overview

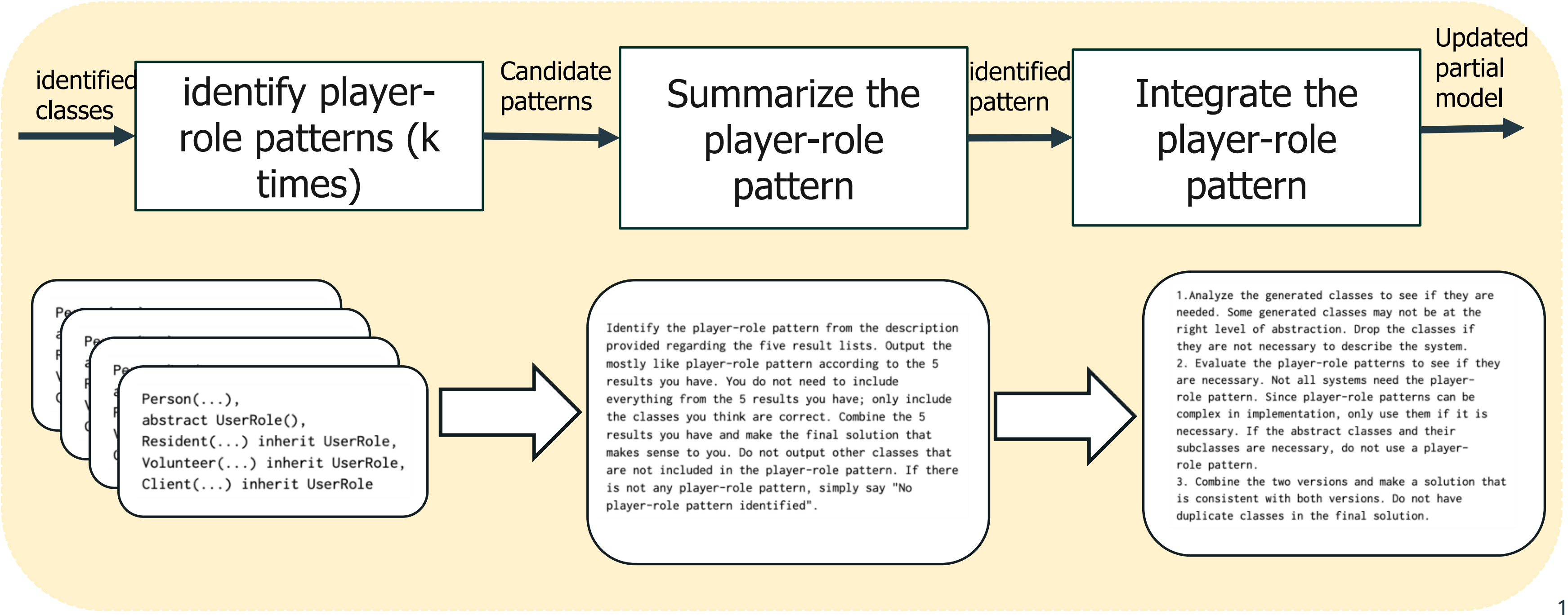

Description
Format description
Few-shot examples
Identify classes and attributes
Identify player-role patterns
Self-reflection
Identify Relation
Complete domain model

Step 1. Identify Classes and Attributes

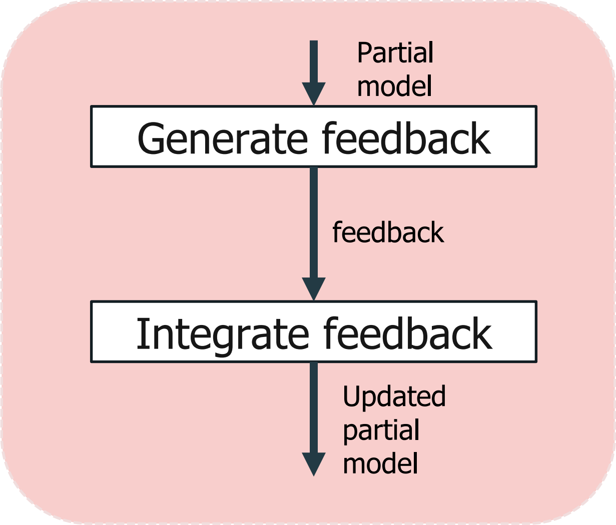
Example: A resident enters a name, street address, phone number, optional email address



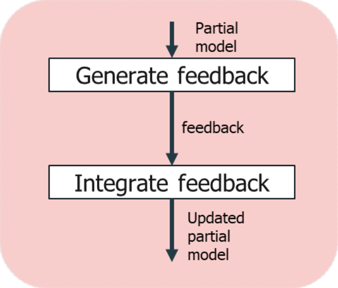
Identify Patterns



Self Reflection



Self Reflection



Duplicated elements

Unnecessary elements

```
Game(string gameStatus, int currentPlayer)
BoardGame(int numberOfPlayers, string gameStatus) inherit Game
Player(string playerName, string playerColor, int playerPosition)
Tile(int tilePosition, string tileColor, string tileType)
Board(int boardSize, string layout)
PlayingPiece(string pieceColor, int piecePosition)
Die(int dieNumber)
ConnectionPiece(int piecePosition)
HiddenTile(int tilePosition) inherit Tile
StartingPosition(int position)
ActionTile(int tilePosition, string actionType) inherit Tile
Deck(int numberOfCards)
ActionCard(string actionType)
Turn(int turnNumber, string playerTurn)
Color(string colorType)
Instruction(string actionInstruction)
enum PlayerNumber(Player1, Player2, Player3, Player4)
enum ActionType(ExtraTurn, ConnectTiles, RemoveConnection, MovePiece, LoseTurn)
enum TileColor(White, Black)
enum TileType(Regular, Action, Hidden)
abstract GameRole()
DesignerRole(string layoutDesign, int hiddenTilePosition, int startingPositions[],
string actionTileLocations[], string actionCardTypes[]) inherit GameRole()
PlayerRole(string playerName, string playerColor, int playerPosition) inherit GameRole()
```

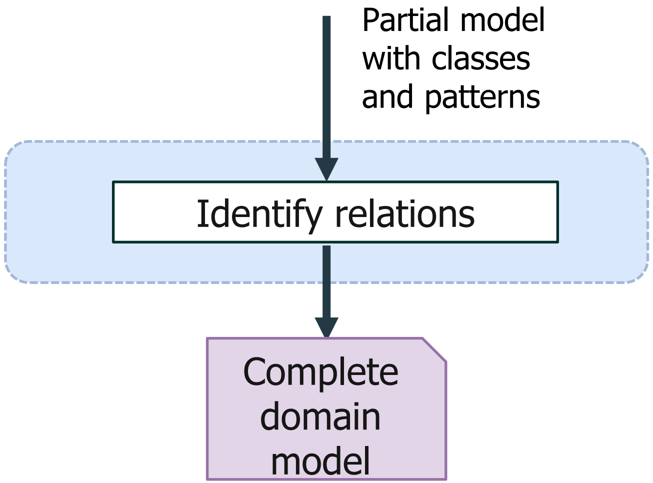
Before self reflection

```
Game(string )
Player(string playerName, string playerColor, int playerPosition)
Tile(int tilePosition, string tileColor, string tileType)
Board(int boardSize, string layout)
PlayingPiece(string pieceColor, int piecePosition)
ile
tionType) inherit Tile
Deck(int numberOfCards)
ActionCard(string actionType)
Turn(int turnNumber, string playerTurn)

enum PlayerNumber(Player1, Player2, Player3, Player4)
enum ActionType(ExtraTurn, ConnectTiles, RemoveConnection, MovePiece, LoseTurn)
enum TileColor(White, Black)
enum TileType(Regular, Action, Hidden)

DesignerRole(string layoutDesign, int hiddenTilePosition, int startingPositions[], string
actionTileLocations[], string actionCardTypes[])
```

After self reflection

Identify Relations

- Composition
  - E.g., 1 H2S contain * Person
- Generalization
  - E.g., Resident inherit UserRole
- Association
  - E.g., 1 Resident associate * Item



Overview

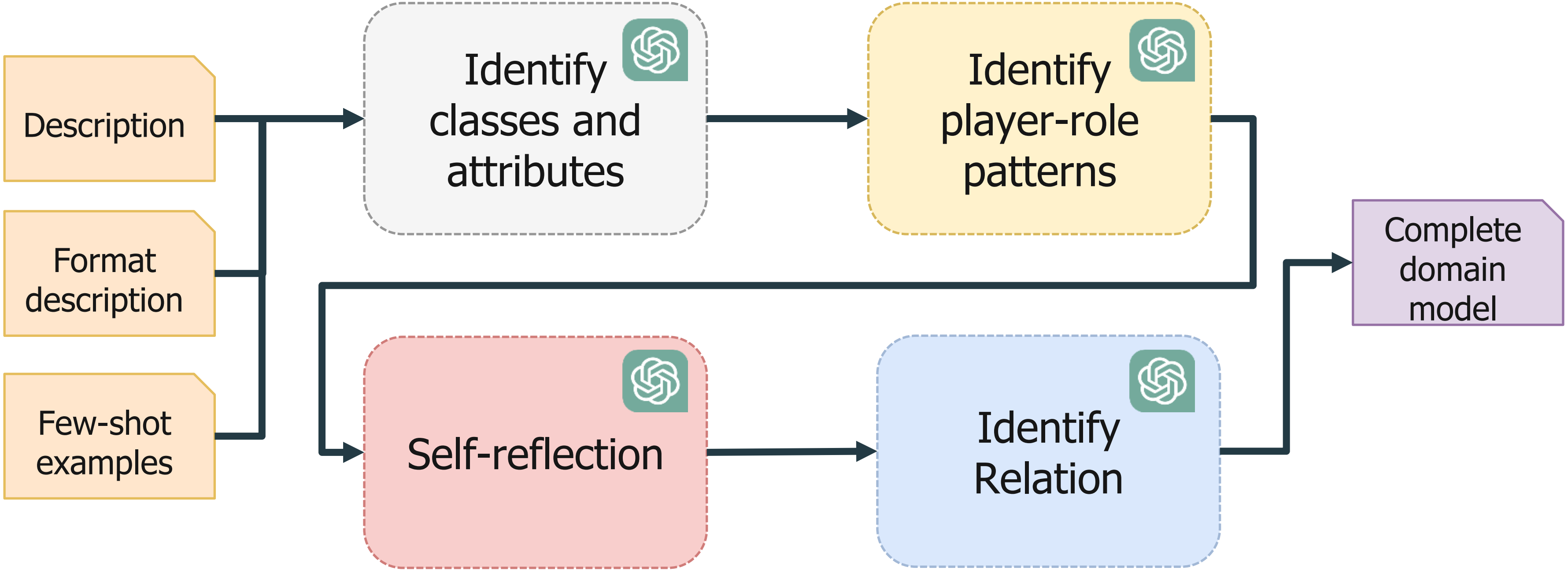

Description
Format description
Few-shot examples
Identify classes and attributes
Identify player-role patterns
Self-reflection
Identify Relation
Complete domain model

Table of Contents

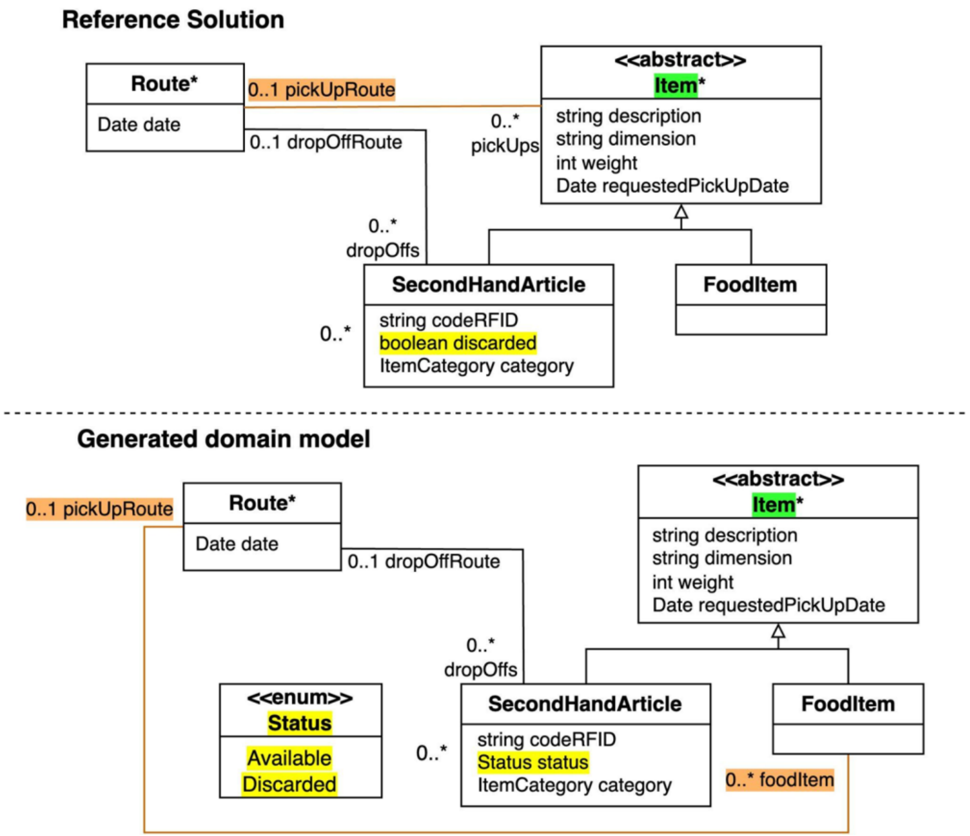
Evaluation Scheme

Case 1: Direct match
Score: 1

Case 2: Semantically equivalent
Score: 1

Case 3: Partial match
Score: 0.5

Case 4: No match
Score: 0



Benchmark and Experiment Setup

The LLM (GPT-4) and configuration for the LLM is fixed for all settings

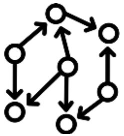
Compared with the single-step approach [7]

A benchmark with diverse set of description-domain model pairs with different complexities [7]

Name	LabTracker	CelO	TeamSports	SHAS	OTS	Block	Tile-O	HBMS
Domain	Medical	Social	Sports	Smart Home	Education	Game	Game	Management
# of classes	16	13	16	23	16	15	18	18
# of attributes	43	23	24	26	25	30	19	32
# of relationships	22	22	20	27	19	24	21	22

Experiment: Research Questions

RQ1: What is the performance of our multi-step LLM-based automated domain model approach compared to a single-step approach?



RQ2: What is the performance of identifying player-role patterns with our multi-step LLM-based automated domain modeling system?

RQ1. Generation Performance

Model Element	Single-step Approach Precision	Single-step Approach Recall	Single-step Approach F_1-score	MIG Approach Precision	MIG Approach Recall	MIG Approach F_1-score
Class	**0.8483**	0.5003	0.6280	0.8021 ↓	**0.7502** ↑	**0.7706** ★
Attribute	**0.5626**	0.5329	**0.5403**	0.4825 ↓	**0.5732** ↑	0.5176 ~
Relationship	0.2867	0.1420	0.1781	**0.3256**	**0.3027** ↑	**0.3120** ★

Compared to the single-step approach, for MIG

- ↑ Recall for all elements ***increases***
- ↓ Precision for classes and attributes *drops slightly*
- ~ Overall F1 score for attributes stays *similar*
- ★ Overall F1 score for classes and relations ***increases significantly***

Player-role Pattern

Approach	Precision	Recall	F_1-score
Zero-shot Single-step	0.9242	0.6143	0.7380
Two-shot Single-step	**1**	0.6571	0.7931
Multi-step	0.83	**0.8**	**0.8147**

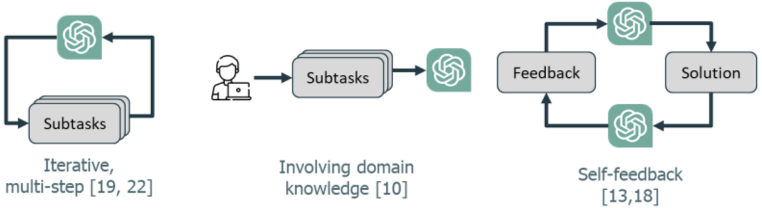
Compared to the single-step approach
- The MIG approach improves the recall significantly
- In MIG LLMs seem to identify more patterns unnecessarily

Table of Contents

Challenge: Domain Model Creation

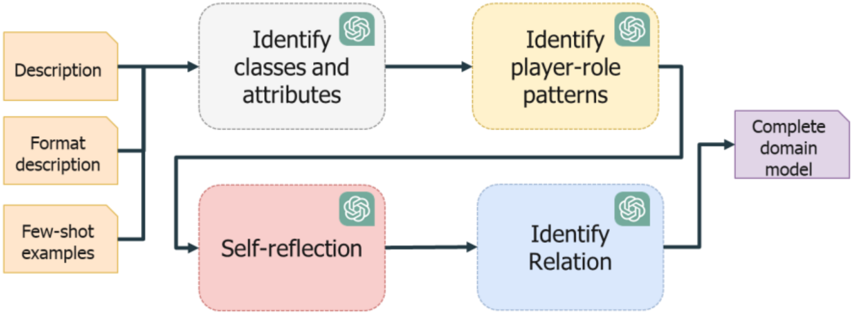
- **Performed manually by software engineers**
  - Time consuming
  - High variation
- **Existing automated approaches**
  - Requires some level of human interaction
  - Domain elements are extracted mostly at sentence level

Practice from LLM Research



Reference numbers are from the paper

Overview



RQ1. Generation Performance

	Single-step Approach			MIG Approach		
Model Element	Precision	Recall	F_1-score	Precision	Recall	F_1-score
Class	**0.8483**	0.5003	0.6280	0.8021 ↓	**0.7502** ↑	**0.7706** ★
Attribute	**0.5626**	0.5329	**0.5403**	0.4825 ↓	**0.5732** ↑	0.5176 ~
Relationship	0.2867	0.1420	0.1781	**0.3256**	**0.3027** ↑	**0.3120** ★

Compared to the single-step approach, for MIG
- ↑ Recall for all elements *increases*
- ↓ Precision for classes and attributes *drops slightly*
- ~ Overall F1 score for attributes stays *similar*
- ★ Overall F1 score for classes and relations *increases significantly*