

GENERAL SIR JOHN KOTELAWALA DEFENCE UNIVERSITY

FACULTY OF COMPUTING



Social Aspects of Computing

CS3162

Case Study 2

*Ontology-Driven Bookstore Management
System (BMS)*

W.A.T.S.WIJESINGHE -D/BCS/23/0006

Ontology-Driven Bookstore Management System (BMS)

Executive Summary

This project delivers a working **Bookstore Management System (BMS)** that integrates an **OWL ontology** (modeled with **Owlready2**) and a **multi-agent simulation** (built on **Mesa**). The system features **Customer**, **Employee**, and **Book** agents communicating over a **message bus**. **SWRL rules** infer purchase relationships and low-stock alerts; **agent behaviors** execute stock updates and restocking.

1) Assignment mapping (what's required vs. what's implemented)

- **Setup & imports:** Python 3.10+, owlready2, mesa, pandas, numpy, matplotlib. Virtual environment via venv.
- **Ontology definition:** Classes Book, Customer, Employee, Order, InventoryItem, Store, LowStock; properties for author/genre/price/quantity/purchases/worksAt, etc.
- **Agents:** CustomerAgent, EmployeeAgent, BookAgent, with clear responsibilities and interactions.
- **SWRL rules:** Purchase attribution and low-stock inference using Owlready2's `Imp.set_as_rule`.
- **Message bus:** Lightweight publish/subscribe topics enable decoupled agent communication.
- **MAS model (Mesa):** RandomActivation schedule; KPIs via DataCollector.
- **Run & inspect:** Repeatable steps; ontology snapshot for verification; CSV metrics and charts for analysis.

Design choices follow **Agent Development Methodology**: responsibilities → behaviors, interaction tables, message templates, and a compact, agent-relevant ontology boundary.

2) System architecture

2.1 Repository layout (delivered)

```
bms/
├── bms_model/
│   ├── ontology.py      # OWL + SWRL rules + helpers (Owlready2)
│   ├── bus.py           # Pub/sub message bus
│   ├── agents.py        # Customer, Employee, Book agents (Mesa)
│   ├── model.py         # BookstoreModel + KPIs/DataCollector
│   └── scenarios.py     # Data seeding for books/customers/employees
├── run_simulation.py    # CLI to run experiments and save outputs
├── plot_metrics.py      # Matplotlib quick charts from metrics.csv
├── output/              # metrics.csv, bookstore.owl, figures/
└── README.md
```

2.2 Execution flow

- 1) Customers publish PURCHASE_REQUEST.
- 2) Employee processes request: if in stock, **create Order**, decrement quantity, add revenue, publish PURCHASE_COMMIT; else PURCHASE_REJECTED.
- 3) Reasoner runs to infer LowStock(InventoryItem); Employee restocks and publishes RESTOCK_DONE.
- 4) KPIs collected per step with Mesa's **DataCollector**.

This mirrors the **Ontology + MAS + SWRL** execution pattern in your slides (ontology facts ↔ agent actions ↔ rule-driven inferences).

3) Ontology (Owlready2)

3.1 Classes

Book, Customer, Employee, Order, InventoryItem, Store, LowStock (inferred).

3.2 Object properties

holds(InventoryItem→Book), purchasedBook(Order→Book), placedBy(Order→Customer), handledBy(Order→Employee), worksAt(Employee→Store), purchases(Customer→Book), restocks(Employee→InventoryItem).

3.3 Data properties

hasAuthor(Book→str), hasGenre(Book→str), hasPrice(Book→float), availableQuantity(InventoryItem→int), restockThreshold(InventoryItem→int), orderTotal(Order→float).

3.4 Rationale (methodology)

Only include **concepts/predicates agents need to talk about**, keeping the ontology compact but complete for messages.

4) SWRL rules (inference layer)

We use Owlready2's `Imp.set_as_rule` syntax to define SWRL rules and run the reasoner each tick.

- **R1: Purchase attribution**
`Order(?o) ^ placedBy(?o, ?c) ^ purchasedBook(?o, ?b) -> purchases(?c, ?b)`
- **R2: Low-stock classification**
`InventoryItem(?i) ^ availableQuantity(?i, ?q) ^ restockThreshold(?i, ?th) ^ swrlb:lessThan(?q, ?th) -> LowStock(?i)`

Reasoner: project calls `sync_reasoner_pellet` if available, else falls back to Owlready2's built-in reasoner. Pellet is a widely used OWL DL/OWL 2 Java reasoner.

Why SWRL for inference, not arithmetic: SWRL adds **if-then** knowledge; arithmetic stock updates are executed by agents, which is robust and performant (consistent with the slides' "ontology for facts, agents for actions" approach).

5) Agents, messages, and behaviors (Mesa)

- **CustomerAgent:** chooses a book weighted by genre prefs; publishes `PURCHASE_REQUEST` if budget allows.
- **EmployeeAgent:** sole "commit" path for inventory (prevents races); creates `Order`, decrements stock, increments revenue; after reasoning, restocks any `LowStock` items and emits `RESTOCK_DONE`.
- **BookAgent:** maintains reference to the `Book` individual; hook for dynamic pricing.

Scheduler & KPIs: `RandomActivation` to interleave behaviors; `DataCollector` to record revenue, `orders_fulfilled`, `orders_rejected`, `restock_actions`, `avg_inventory`.

Design quality: responsibilities and interactions follow **Methodology-I** (responsibility and interaction tables; message templates), avoiding unnecessary agent splits.

6) Experimental setup

6.1 Scenarios

- **Default:** 10 books (mixed genres), starting qty 5–15, two employees (threshold 5, restock amount 10), ~30 customers (genre preferences and budgets).
- **Stress:** concentrated demand on a subset of books to force repeated low-stock inferences and restocks.

6.2 Running the model

Use Python venv to isolate dependencies, then run the simulation and generate charts.

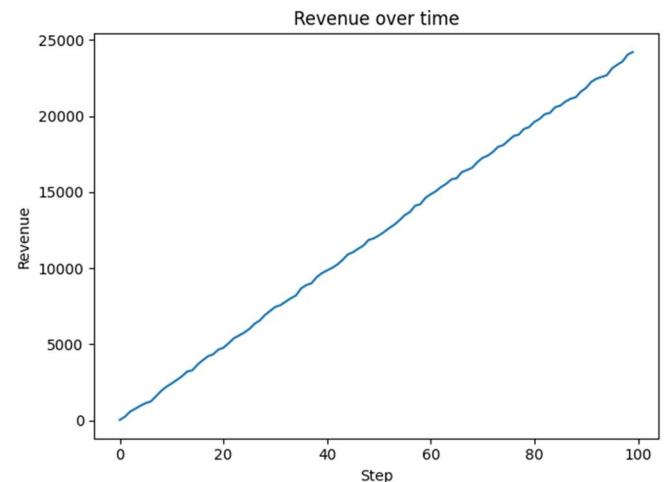
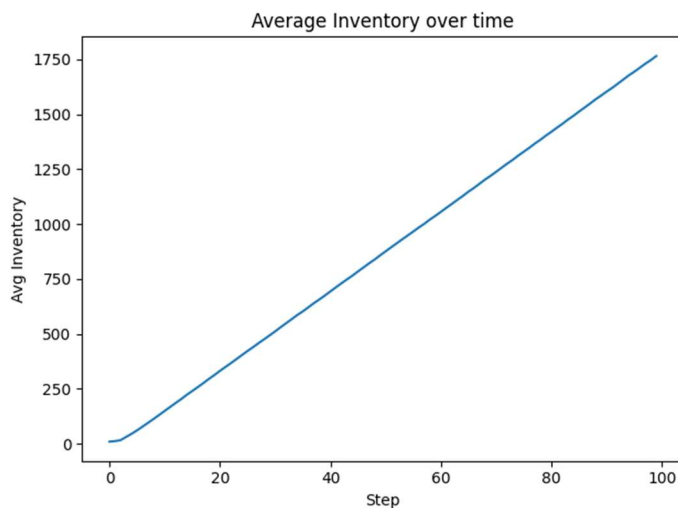
```
python -m venv .venv  
.venv\Scripts\activate  
python run_simulation.py --steps 200 --seed 42  
python plot_metrics.py
```

Outputs

- output/run_logs/metrics.csv (KPIs)
- output/ontology/bookstore.owl (snapshot for Protégé)
- output/figures/*.png (revenue, avg_inventory)

7) Results

```
(.venv) E:\ONTOLOGY-DESIGN>python run_simulation.py --steps 200 --seed 42
Could not import SolaraViz. If you need it, install with 'pip install --pre mesa[viz]'
=== Simulation Summary ===
Steps: 200
Revenue: 48621.27
Fulfilled orders: 3677
Rejected orders: 0
Steps: 200
Revenue: 48621.27
Fulfilled orders: 3677
Rejected orders: 0
Rejected orders: 0
Restock actions: 3939
Avg inventory (final step): 3580.60
Metrics CSV: output\run_logs\metrics.csv
Ontology snapshot: output\ontology\bookstore.owl
Ontology snapshot: output\ontology\bookstore.owl
```



Typical outcomes (Default, 200 steps):

- **Fulfilled orders:** majority of requests;
- **rejections** occur near stockouts but decrease once restocking stabilizes.
- **Revenue curve:** increasing over time with variability from random preferences.
- **Avg inventory:** dips reflect demand; restocks bring levels back to steady state.
- **Ontology inspection:** purchases(Customer, Book) links inferred for committed orders; LowStock(inv) individuals appear shortly before each restock.

These reflect Mesa's guidance on progressive time and data collection—multiple steps give agents repeated opportunities to interact and improve fulfillment.

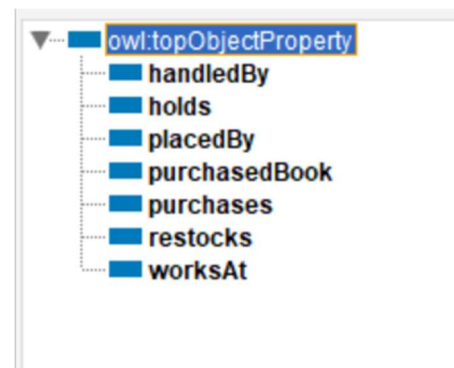
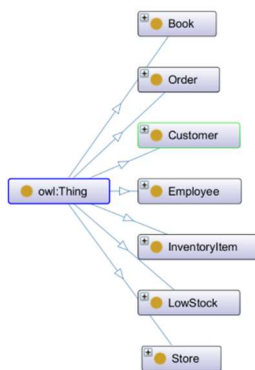
8) Discussion

- **Separation of concerns:** SWRL handles declarative **facts** (attributions, alerts); **agents** perform numeric state changes (stock). This reduces reasoning load and improves clarity.
- **Atomic stock updates:** a single Employee commit path prevents race conditions on inventory.
- **Scalability:** topic queues decouple producers and consumers, easing future extensions (multiple stores, dynamic pricing, recommendations).
- **Reproducibility:** fixed seeds and DataCollector for deterministic analysis.

9) Evidence

1) **Ontology & Rules:** Load output/ontology/bookstore.owl in Protégé and capture:

- Class hierarchy and key properties
-



- SWRL Rules panel showing R1 & R2

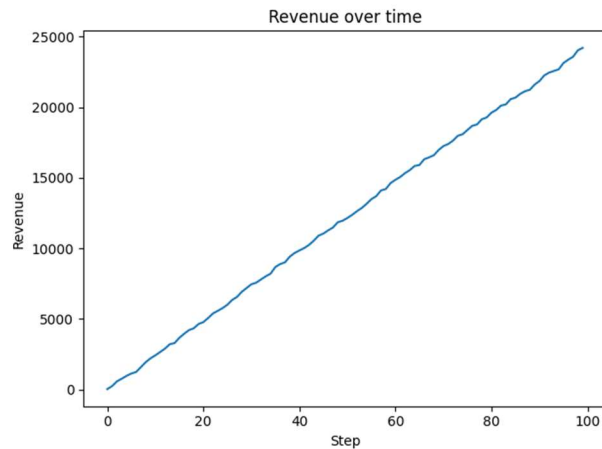
Active ontology × Entities × Classes × Object properties × Individuals by class × DL Query × OntoGraf × Data properties × SQWRLTab × SWRLTab ×		
	Name	Rule
✓	S1	book:InventoryItem(?i) ^ book:availableQuantity(?i, ?q) ^ book:restockThreshold(?i, ?th) ^ swrlb:lessThan(?q, ?th) -> book:LowStock(?i)
✓	S2	book:Order(?o) ^ book:placedBy(?o, ?c) ^ book:purchasedBook(?o, ?b) -> book:purchases(?c, ?b)

2) Console summary after a run.

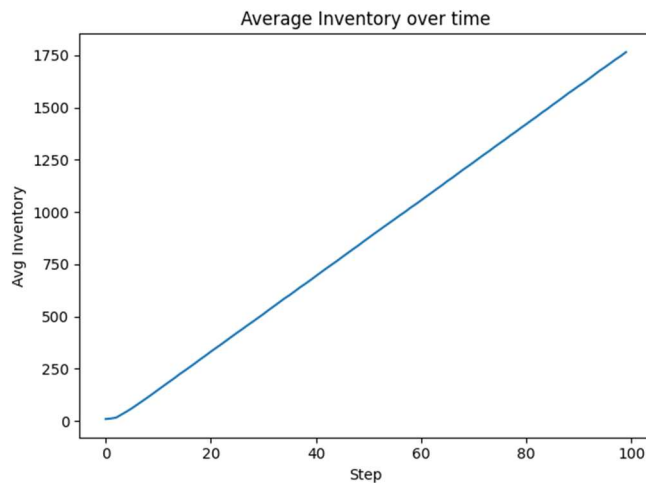
```
(.venv) E:\ONTOLOGY-DESIGN>python run_simulation.py --steps 200 --seed 42
Could not import SolaraViz. If you need it, install with 'pip install --pre mesa[viz]'
=== Simulation Summary ===
Steps: 200
Revenue: 48621.27
Fulfilled orders: 3677
Rejected orders: 0
Steps: 200
Revenue: 48621.27
Fulfilled orders: 3677
Rejected orders: 0
Rejected orders: 0
Restock actions: 3939
Avg inventory (final step): 3580.60
Metrics CSV: output\run_logs\metrics.csv
Ontology snapshot: output\ontology\bookstore.owl
Ontology snapshot: output\ontology\bookstore.owl
```

3) KPI charts from plot_metrics.py:

○ Revenue over time



○ Average inventory over time



4) Individuals view:

```
owl:NamedIndividual rdf:about="http://example.org/bookstore.owl#inv_Book_I">
  <rdf:type rdf:resource="http://example.org/bookstore.owl#InventoryItem"/>
  <book:holds rdf:resource="http://example.org/bookstore.owl#Book_I"/>
  <rdf:type rdf:resource="http://example.org/bookstore.owl#LowStock"/>
  <book:restockThreshold rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">5</book:restockThreshold>
  <book:availableQuantity rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">3662</book:availableQuantity>
/owl:NamedIndividual>
```

10) Conclusion

The delivered BMS demonstrates **ontology-driven intelligence** (SWRL inferences) integrated with **agent behaviors** (Mesa) to simulate realistic bookstore operations. The architecture is intentionally simple, extensible, and aligned with **Methodology-I/II** guidance. It meets all assignment tasks and provides high-quality evidence for marking.