Linnéuniversitetet
Kalmar Växjö

Group report

Optimized Retail Restocking

Författare: Razan Kngo, Rula Nayf, Amala Antony
Handledare: Lars Karlsson
Examinator: Diego Perez Palacin
Termin: HT24
Kurskod: 4DT903

Table of contents 2

Introduction

Efficient retail restocking plays a pivotal role in maintaining customer satisfaction and ensuring sustainable business operations. Retailers often face challenges in managing inventory and logistics, such as delayed restocking, stockouts, and inefficient resource utilization. These challenges can lead to lost sales opportunities, customer dissatisfaction, and increased operational costs. Addressing these issues requires a cohesive system that integrates inventory management with logistics to provide a seamless flow of goods from warehouses to retail shelves.

The *"Optimized Retail Restocking through Integrated Inventory and Logistics Management"* project presents an innovative solution to these challenges. This application combines the domains Retail Inventory Management and Logistics Management domains, utilizing advanced technologies and principles of Model-Driven Software Engineering (MDSE). By automating key processes such as order generation, delivery scheduling, and route optimization, the system minimizes manual intervention and reduces errors while enhancing overall efficiency and scalability.

One of the primary objectives of the project is to ensure timely and efficient restocking of retail stores. By leveraging distance matrix data and delivery constraints, the system generates optimized delivery schedules that consider factors like vehicle capacities, delivery windows, and route constraints. This ensures minimal delays, reduced travel costs, and optimal use of resources. Additionally, the system's modular design allows for scalability and adaptability to meet the evolving needs of the retail sector.

The project not only addresses existing inefficiencies but also demonstrates the potential of MDSE to solve complex, real-world problems. By providing a structured, automated, and scalable solution, the *"Optimized Retail Restocking"* application represents a significant advancement in modern retail management. It empowers businesses to enhance operational efficiency, improve customer satisfaction, and maintain a competitive edge in the market.

Overview

The "Optimized Retail Restocking" application is designed as a comprehensive system to streamline retail inventory and logistics processes. Here's an overview of how it works:

How It Works:

The system begins by capturing retail store-specific data, including inventory levels and delivery requirements, in the Retail Store Model (MM1), while vehicle information is stored in the Fleet Information Model (MM3). Automated workflows

use Model-to-Model (M2M) transformations to generate restocking orders from inventory data and combine them with fleet details to define delivery constraints (MM4). External tools like Maps APIs generate travel time and distance data (MM5), which are processed by a routing optimization tool to create efficient delivery routes (MM6). These routes are visualized in an easy-to-use format for logistics managers, ensuring timely and cost-effective restocking. Any updates in inventory or fleet data trigger dynamic adjustments across all models, maintaining consistency and accuracy in operations.

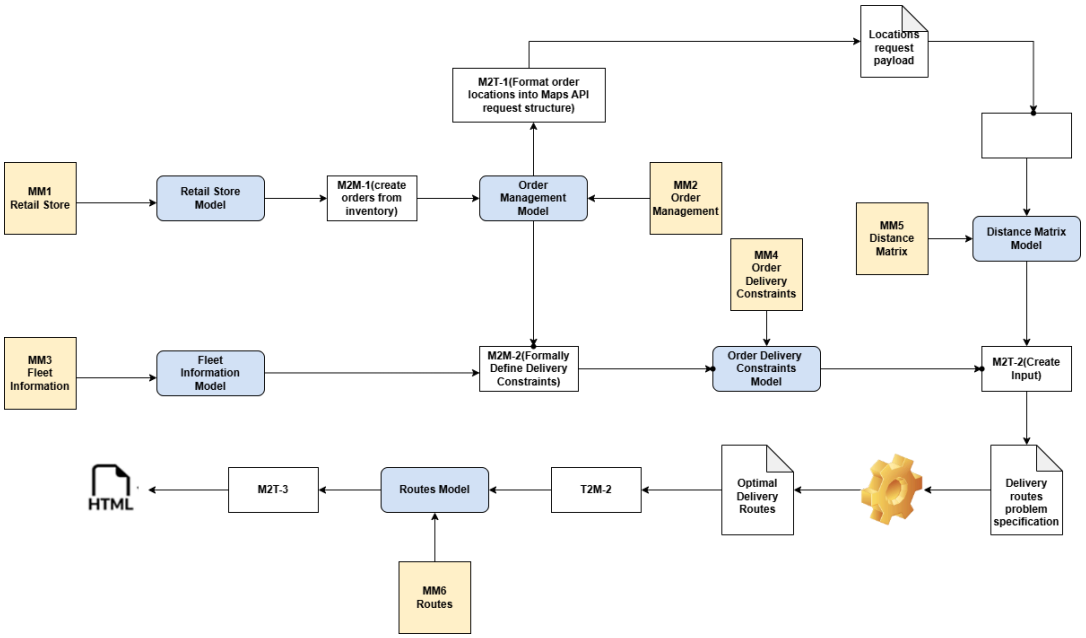
Key Features:

- **Automation:** The application minimizes manual effort by automating processes such as order generation, delivery scheduling, and routing. Model-to-model (M2M) transformations ensure seamless data flow and reduce human error.
- **Seamless Integration:** By integrating inventory and logistics data, the system ensures a unified approach to restocking, enabling smooth interoperability between components like Maps APIs and routing optimization tools.
- **Scalability:** Designed with a modular structure, the system can easily accommodate new functionalities, expand to additional retail locations, or integrate with future technologies, making it adaptable to changing business needs.
- **Enhanced Efficiency:** The system optimizes resource utilization by generating the most efficient delivery schedules and routes, minimizing travel costs and reducing delays.
- **Dynamic Updates:** Real-time updates in inventory levels or vehicle availability automatically propagate through the system, ensuring all dependent models reflect current data for accurate decision-making.
- **User-Friendly Visualization:** The application provides clear, structured visualizations of optimized routes, allowing logistics managers to plan and execute deliveries effectively.
- **Cost-Effectiveness:** By optimizing delivery operations and reducing wasted resources, the system contributes to significant cost savings for businesses.

This streamlined process ensures timely and efficient retail restocking, improving overall operational performance and customer satisfaction.

MDE system architecture

Our Model-Driven Engineering (MDE) system architecture is designed to optimize retail restocking through seamless integration of inventory and logistics management. It leverages models, metamodels, and transformations to automate complex workflows while maintaining data consistency and flexibility.



Key Components of the System

1. **Metamodels and Models (MM1- MM6)**

   The Retail Store Model (MM1) serves as the foundation for all further processing by representing raw, unstructured data from retail stores. It captures essential information like store locations, items, and delivery windows, providing the basic data needed to identify which stores require restocking.

   The second one is the Order Management Model (MM2), which is built on MM1 by organizing this raw data into structured orders. It includes critical details such as order IDs, the items being ordered, their quantities, delivery windows, and delivery locations. This structure ensures that only stores needing restocking are considered and that their needs are clearly defined for logistics.

   The Fleet Information Model (MM3) focuses on the resources available for deliveries. It stores details about delivery vehicles, including their types, capacities, availability, and any unique characteristics, making it a key element in planning logistics efficiently.

   The Order Delivery Constraints Model (MM4) brings everything together by integrating the structured order data from MM2 with the fleet details in MM3.

This model defines the constraints that influence delivery operations, such as vehicle capacity limits, the number of vehicles available, delivery windows, and required delivery times for each order.

The Distance Matrix Model (MM5) plays a key role in planning efficient delivery routes. It contains calculated distances and travel times between locations, providing the necessary spatial and timing details for effective route optimization.

Finally, the Routes Model (MM6) represents the optimized delivery routes created using MM4 and MM5. It provides a structured view of the best paths for delivery operations, ensuring that logistics planning is as effective as possible.

2. Transformations

Two Model-to-Model (M2M) Transformations:

1. The first is used to convert retail data from MM1 into structured orders, (MM1 → MM2).
2. While the second model-to-model transformation takes two inputs and gives an output. This transformation combines order and fleet data to define delivery constraints,
   (MM2 + MM3 → MM4).

Three Model-to-Text (M2T) Transformations

1. MM2 → JSON, the first M2T is responsible for converting order data for use in distance and routing calculations.
2. MM4 + MM5 → Routing Tool Input, while the second M2T prepares data for route optimization.
3. MM6 → HTML, lastly, we use M2T transformation to generate human-readable visualizations of delivery routes.

Two Text-to-Model (T2M) Transformations:

1. Generated distances → MM5. The first Text-to-Model transformation processes programmatically generated distances into a structured model (MM5).

2. Routing Tool Output → MM6: The other text-to-model processes optimized route data into the Routes Model.

3. Tools

The tools used in this project are essential for creating, transforming, and managing models effectively, ensuring seamless integration and optimization of the workflow.

For creating and managing metamodels (e.g., MM1, MM2, MM3) **Eclipse** Modeling Framework (EMF) was used. While for defining M2M transformations, QVTo was used, ensuring precise mappings between models. To handle M2T transformations we used Accelo, generating formats like JSON for further integrations. While custom Java

programs were used to implement T2M transformations, including distance calculations and route optimizations.

Workflow Overview
The system follows a sequence of transformations:

First we started with data preparation, where MM1 was manually populated with retail data, which is transformed into MM2 using M2M transformations. This ensures only stores requiring restocking are included. Then we did step for integration of fleet data, since MM3 was also manually created, and then using a M2M transformation we merged MM2 and MM3 into MM4, capturing delivery constraints.

Route Optimization
MM4 is enhanced with distance data from MM5 that is generated through a T2M transformation using programmatically calculated distances. This combined information is used to prepare inputs for an external routing tool, which calculates optimized routes. The results are then converted into MM6, and for easy review by stakeholders, MM6 is transformed into an HTML format for clear visualization.

MDSE problems and solutions

At the beginning of the project, one of the initial challenges was the lack of clarity about the complete structure and final goals of the system. This uncertainty made it difficult to fully visualize how the different components and transformations would fit together. As the project progressed and individual transformations were developed, the bigger picture gradually became clearer. This iterative refinement allowed us to align the individual elements effectively and ensure a cohesive system design.

In the early stages, the system lacked an automated process for restocking retail stores and planning deliveries. The raw and unstructured inventory data in MM1 made it difficult to generate actionable order plans and delivery schedules. We implemented an M2M transformation (MM1 to MM2) using ModeltoModel1.qvto. This transformation converted raw inventory data in MM1 into structured orders in MM2, including details like item quantities, delivery locations, and time windows. This structured output enabled efficient delivery planning by providing clear, actionable inputs for logistics optimization. Additionally, the orders generated from MM2 served as the foundation for integrating fleet data in subsequent transformations, completing the restocking and delivery planning process.

One of the most significant and time-consuming challenges was the limitation of Acceleo, which does not support multiple input models in a single run. Since our project required combining data from multiple input models (e.g., MM4Output.xmi and DistanceMatrix.xmi), we resolved this issue by processing each input model separately and programmatically combining their outputs within the Generate.java file. This

additional step increased the complexity of the workflow but ensured that the final output met the requirements.

Another challenge was related to the integration of external tools. Initially, we planned to integrate a routing tool to calculate optimal delivery routes. However, due to the complexity of integration, we implemented Dijkstra's Algorithm in Java. This algorithm enabled us to compute the shortest paths for each location, effectively providing a route optimization solution. Similarly, the planned integration with the Maps API to create a Distance Matrix Model (MM5) for routing was removed because of time and technical constraints. This left the system without a mechanism to calculate distances and travel times between delivery locations. A Text-to-Model (T2M) transformation was developed to populate MM5. Using a Java-based simulation, distances and travel times were programmatically generated for each pair of locations. Two distances and travel times were calculated for each pair to account for variability in delivery routes. This approach ensured flexibility in route planning while eliminating the need for Maps API integration.

Additionally, during the project we faced challenges dealing with error detection in transformations. Which often caused issues that impacted the next stages of the system. Examples of these errors included missing incorrect data formats especially in JSON files, and mismatches between the expected and actual output structure. Detecting these errors early was crucial to ensure the smooth working of the system and avoid unnecessary delays. To solve it, we added a validation function in the Generate.java file. This function checked the generated JSON files for common issues before moving forward. If an error was found, the system flagged it immediately and provided detailed feedback so we could fix it quickly. This ensured that all the necessary fields were present and correctly formatted.

Improvements and extensions

While the current Model-Driven Engineering (MDE) system provides an effective framework for optimizing retail restocking and logistics, there are several areas for improvement and potential extensions to enhance its flexibility, reliability, and scalability.

Proposed Improvements

Improved tooling

The system uses Acceleo for Model-to-Text (M2T) transformations. This tool has limitations, such as only supporting single input models at a time. This increases reliance on custom Java code to merge outputs, which can be time-consuming to maintain. Alternative tools that support multiple input models could simplify workflows and improve efficiency.

Additionally, while we initially planned to integrate the Maps API for generating real-time distance matrices, this feature was replaced by a manually created distance

matrix. Revisiting this integration with a routing API or pre-processed data from tools could allow for more accurate and dynamic distance updates in the future.

Real-Time updates
To make the system more adaptive, event-driven models could be introduced to trigger updates automatically whenever key inputs, such as MM1 (store data) or MM3 (fleet data), are modified. This could eliminate direct handling and ensure that dependent models remain up to date.

Error handling
Improving error detection and handling is a critical area. Currently, issues during some transformations have been addressed manually. Adding validation pipelines could catch such errors earlier while automating error logging and reporting would make debugging faster and reduce the likelihood of failures throughout the workflow. To enhance error detection further, we propose using tools similar to those for metamodel validation in Ecore. These tools could automatically check if the generated JSON or XMI files match their expected structures, making it easier to catch problems early and ensure consistency throughout the system.

Extensions to the System
Dynamic decision
The current system uses static data for route planning and distance calculations. Introducing real-time adaptability, such as adjusting delivery plans based on live traffic or weather conditions, could further improve delivery efficiency. This could be achieved by integrating external APIs that provide dynamic data. Machine learning models could be introduced to learn from past delivery patterns, refining route optimization and decision-making over time. By analyzing historical and real-time data, these models can predict the best routes, adjust to changing conditions like traffic or weather, and improve delivery time estimates.

Improved visualization
The system currently generates an HTML output of the optimized routes (MM6) for stakeholders. This could be further developed into an interactive app, featuring dynamic maps and charts for more intuitive route analysis.

Conclusion

The "Optimized Retail Restocking" project highlights the power of Model-Driven Software Engineering (MDSE) in solving complex, real-world problems in the retail and logistics domains. By employing a combination of models, metamodels, and transformations, the project successfully addressed key challenges, such as handling unstructured data, integrating diverse data sources, optimizing delivery routes, and generating user-friendly outputs.

Instruction

Setup

- Install Java JDK version 11 or higher
- Eclipse IDE with the following plugins:
    - Eclipse Modeling Framework (EMF) for metamodels
    - QVT operations for model-to-model transformation
    - Acceleo for model-to-text transformation
- JSON library since it included in the project

Execution Steps

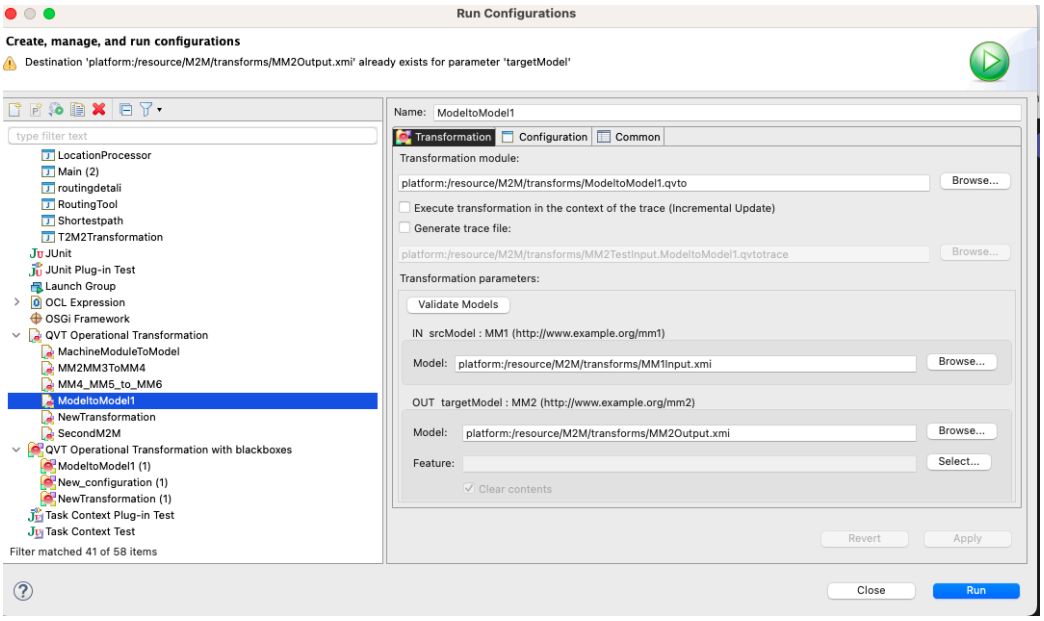
Needed Metamodels: Each metamodel required for the project is in its respective folder within the model directory. MM1: Retail store model, MM2: Order model, MM3: Fleet model, MM4: Delivery constraints model, and MM5: Distance Matrix model.

Run Transformation

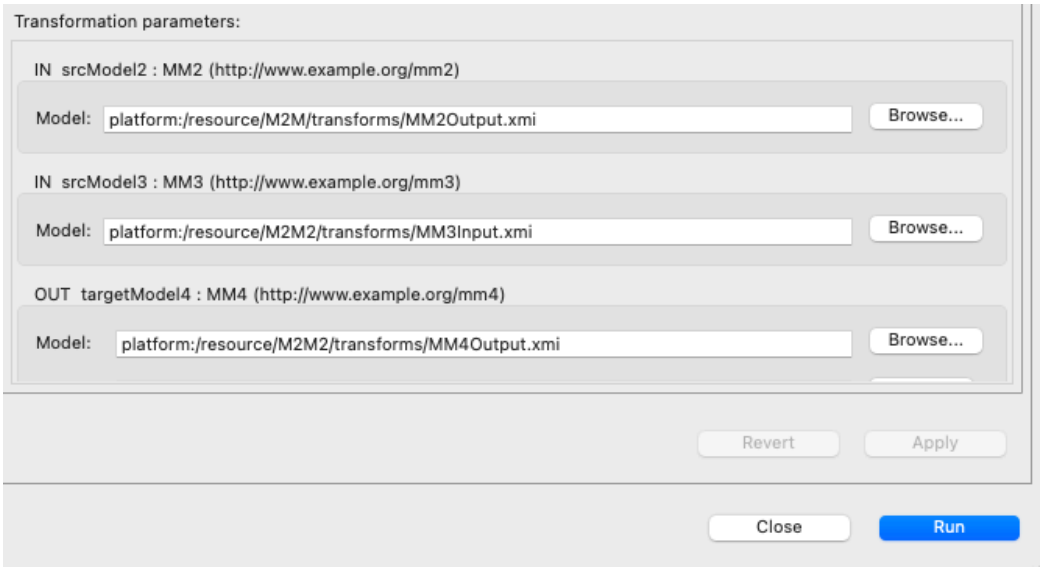
The project implements three types of transformations: M2M, M2T, and T2M as mentioned earlier in the rapport.

1. Model-To-Model transformation:

   M2M-1 in our project is named *M2M* in the transforms folder, Mapping retail stores and their related components in MM1 to structured orders and delivery details in MM2, is done on *ModeltoModel1.qvto*. To run it right click on the file run → run configuration → QVT operational transformation. Select ModeltoModel1.qvto from the list. As you can see in the added screenshot that shows the run configuration, under Transformations parameters select provide two boxes. Input File: The path to MM1Input.xmi (e.g., platform:/resource/MM1Input.xmi). Output File: The desired path for MM2Output.xmi (e.g., platform:/resource/MM2Output.xmi). The requirement is to be an xmi file to map correctly. Then to execute the transformation click on apply then run.



The same steps are for the M2M2, select MM2MM3ToMM4.qvto and here the project has two inputs inserted one so here in the boxes as you can see in the screenshot below.



2. Text-to-Model (T2M) Transformations

MM5 → mm5.tests → DistanceMatrixT2M.java, Right click → run As → java Application → DistanceMatrixT2M
the output will located on /M2T2/tasks/DistanceMatri.xmi
Same for the second T2M

3. Model-to-Text (M2T) Transformations

- Transformation: MM2 → JSON for Maps API

Right-click on Generate.mtl → run → run configuration → Acceleo Application

Select Generate, specify MM2Output.xmi as the input and in the target box choose where you want the output located in our case in the tasks folder so the path will be /M2T/tasks. Output: OrderLocations_MapsAPIRequest.json and OrderSummaryAndContent.json.

- Final step transformation MM6 → HTML

In M2T3 right click on /M2T3/src/M2T3/main/generate.mtl run as → run configuration → Acceleo Application then select Optimal
Input: /T2M2Transformation/src/mm6/RoutesModel.xmi
Target: /M2T3/tasks in the tasks folder locate Routes.html which represents the final output.