ISSN: 2754-6705

Journal of Mathematical & Computer Applications

SCIENTIFIC Research and Community

Review Article

Open Access

Enhancing Event Handling in Warehouse Management Systems through the Adapter Design Pattern: A Structural Approach to Data Integration

Gautham Ram Rajendiran

USA

ABSTRACT

Warehouse Management Systems (WMS) are required for the smooth flow of merchandise from supplier to customer in order to coordinate activities at several touchpoints: from managing inventory, order processing, and shipping. In scaling and evolution, the WMS solution has to integrate with a variety of external and internal systems that often provide their data in different formats and protocols. Often the challenge arises when these systems use heterogeneous sources for data intake, thus making the maintenance of unified data intake and processing difficult.

In this paper, we discuss the solution using the Adapter design pattern. The Adapter pattern will act as a mediator that will convert the data from different sources into one standard format that the WMS is able to handle consistently. It reduces the coupling between core modules of WMS and data sources, thus providing a flexible, extensible, and maintainable system. The paper shall further present an in-depth study of the use of Adapter pattern in handling events within WMS and shall be supported with diagrams for clarity.

***Corresponding author**

Gautham Ram Rajendiran, USA.

Received: August 03, 2022; **Accepted:** August 10, 2022, **Published:** August 17, 2022

Keywords: Warehouse Management System, Event Handling, Adapter Design Pattern, Data Integration, Data Transformation, Event-Driven Architecture

Introduction

The WMS is a core component of logistics and supply chain management, which tracks products from the time they arrive in a warehouse through their final destination. A typical WMS architecture interfaces with other external systems, such as suppliers, third-party logistics providers, and e-commerce platforms, as well as internal systems, like order management systems and inventory databases [1,2].

Given the heterogeneity nature of these systems, the WMS has to handle events coming from multiple different systems. These events can range from inventory and order updates to shipment notifications, just to mention a few, which are often issued in dissimilar formats. This urges a flexible mechanism that can handle data heterogeneity without inducing tight coupling between the core of the WMS and its surrounding dependencies [3].

Problem Statement

Traditional ways of handling data from multiple sources involve developing dedicated handlers for each source. This works in small-scale implementations, but it indeed gets cumbersome and hard to maintain once the number of sources increases. For each new source to be integrated, the code in the WMS has to be changed, which, among other things, makes the architecture brittle.

Proposed Solution

This paper proposes that the Adapter design pattern can be applied for the purpose of regularizing interactions between the WMS and various sources. An Adapter pattern represents a structural design pattern; or more precisely, it acts like a bridge between incompatible interfaces [4]. The Adapter thus changes source-specific formats of the data into one common regular format that the WMS is capable of processing without need to know all different source formats.

Application of the Adapter Pattern to WMS Event Handling

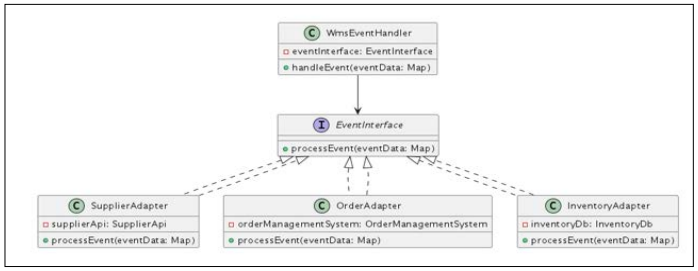

Figure 1: Components

The components shown in Figure 1 are explained below.

Event Interface
The standardized interface that provides the basic abstraction which will define how the events will be processed within the WMS. This interface sets up a contract that should be implemented by every adapter. It therefore allows coherence in the treatments from one source to another. It usually includes a method responsible for the processing of the event, such as processEvent(eventData: Map).

Responsibilities
1. Provides a unified way of processing events.
2. Maps incoming event data from different sources to a standardized format.
3. Serves as the event handler interface of the WMS.

Interactions
Each adapter implements the Event Interface. This interface defines the methods invoked by the WmsEventHandler client for processing the events without regard to the implementation detail of the source systems.

Implementation Considerations
This interface needs to be flexible for most of the event types without frequent changes. An example could be that the parameter eventData can be a generic data structure-like Map or Dictionary for allowing different attributes based on an event type.

Event Handler (Client)
The Event handler is an entry-point into the event-processing in Warehouse Management System. The Event handler is responsible for communicating to the Event Interface for the processing of the events. It decouples from any real source of events, while the transformation and standardization of event data is carried out using adapters.

Responsibilities
1. Receives raw events data from the providing sources.
2. Use the Event Interface to handle each event with the corresponding adapter.
3. Centralizes event handling common logic such as logging, validation, and error handling.

Interactions
The Event Handler will communicate with the Event Interface, implemented by the adapters. It will invoke the processEvent method on that interface, since this forms an uniform way to process events. Depending on the event-type it may also communicate with other modules of the WMS, such as inventory management or order processing.

Implementation Considerations
It has to be thread-safe to cater to a high number of events running concurrently through WmsEventHandler. In case of big volume, it will support async processing requirements imposed by the architecture using message queues or event-driven frameworks like Apache Kafka [5, 6].

Source Adapters
Each of the adapters will implement the Event Interface that is to be put between the WMS with any given external source. Such an adapter would do the job of translation of source-specific event format to standardized one which the WMS understands; doing the job of representation. It will also carry out other source-specific logic like authentication, protocol handling, and data parsing.

The Figure 2 below demonstrates the various adapter states:

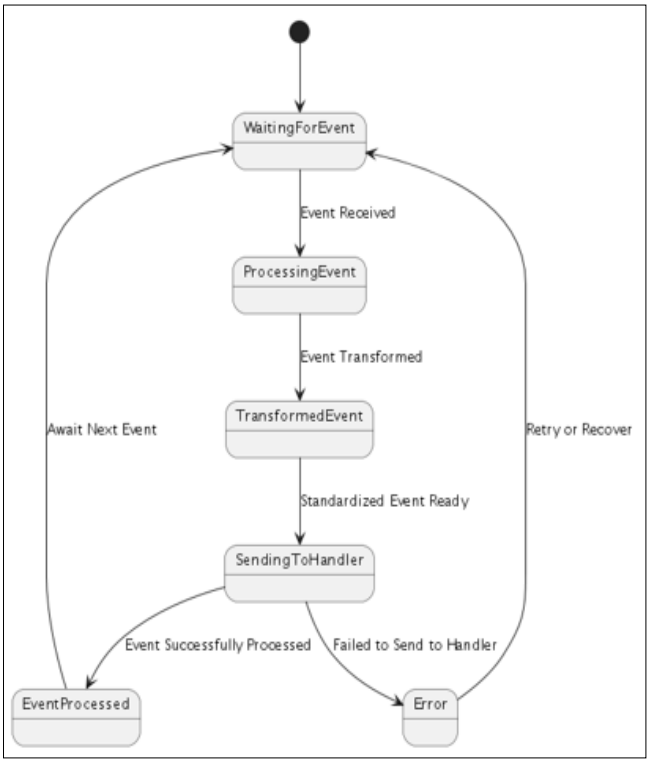

Figure 2: Adapter States

Responsibilities
1. Translate source-specific event formats to the normalized format specified by the Event Interface.
2. Handle source-specific protocols, possibly including REST APIs, file parsing, or interactions with Message Queue.
3. Perform data validation, and optionally add metadata to the event data.

Interactions
The Source Adapters will interact with their respective Source Systems, which are the Adaptees, fetch data, and then transform it. After transformation, the adapter sends the transformed data via the Event interface to WmsEventHandler.

Implementation Considerations
Each Adapter should encapsulate all source-specific logic so that WmsEventHandler remains unaware of the details of source systems. It's allowed that adapters do retries or handle other error situations elegantly if the external systems are unreliable.

Source Systems
The source systems are different external or internal entities that generate events targeting the WMS. They could involve APIs, databases, file systems, and message queues. Each of them has its own data format and communication protocol, abstracted by the respective adapter.

Responsibilities
1. Create events in source format.
2. Expose APIs, files, or messages that may be consumed by adapters for transformation

Interactions
The source systems will interface directly with the adapters. Each of these adapters pulls, pushes, or gets data from the source system and then converts it to the standard event format that WMS requires.

Implementation Considerations
Since source systems are normally outside of the control of the WMS development team, adapters need to be designed taking into consideration all changes or any inconsistencies that would happen in these systems-for example, schema changes or API updates.

External Systems that Interact with the WMS
Event handling in a WMS involves receiving, processing, and responding to events as representations of changes in the state of the warehouse. These can be events that can be used to estimate stock levels, changes to an order, or shipment statuses. These events will originate from a host of sources and more often than not, will be in different formats. A few such systems that act as source of events are elaborated below.

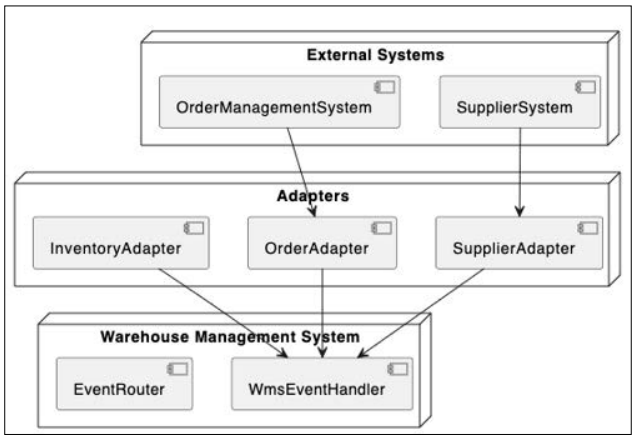

Figure 3: System Interaction

Supplier Systems
These are the external entities that supply the goods or raw materials to the concerned warehouse. Normally they notify about their coming shipment, confirm orders placed and their delivery schedule. These events act like a memo to WMS as to when goods will be available, when to clear space, or whether orders can be delivered on time.

Event Types
1. **Shipment Notices:** This provides notification that shipment is en route. It should include shipment ID, time of expected delivery, and the quantity of items shipped.
2. Order Confirmations Confirmation from a supplier that they have accepted a purchase order. They will contain an order ID, Item details, and Quantity.
3. **Quality Inspection Results:** Details about quality checks carried out on merchandise prior to shipment.

Format and Protocol
Data structures JSON or XML. Web services like RESTful APIs or SOAP.

Challenges
1. Different suppliers may use different data formats (e.g., CSV, JSON, XML), requiring the adapter to handle format conversions.
2. Protocol differences, such as REST vs. SOAP, may necessitate additional libraries or tools within the adapter.

Order Management Systems
Order management systems should provide order management for customers (create, modify, cancel orders). A WMS should be informed regarding the changes in the status of orders in order to correctly allocate the inventory and update its internal view of the current operational plan on how the order fulfillment will happen inside the warehouse.

Event Type
1. **New order placement:** A customer placed a new order that should be allocated by the WMS, preparing for order picking and packing.
2. **Order change of:** Changes in an already existing order; order alteration with regards to the number, article, and delivery address.
3. **Order Canceled:** Notification that the order was canceled and its inventory was not deallocated.

Format and Protocol
JSON or XML payloads. RESTful APIs, message queues, like RabbitMQ, SQS, or even a file system level with CSV or XML.

Challenges
1. OMS systems often have complex schemas that may not map directly to the WMS data model, requiring data transformation and mapping logic.
2. High volumes of order events necessitate the use of asynchronous processing mechanisms to avoid performance bottlenecks.

Inventory Management Systems
The Inventory Management Systems keep track of stock on hand, item locations, and stock movements into and out of the warehouse [7]. It is very important that the WMS should keep in sync with the inventory system to avoid stock discrepancies and incorrect order fulfillment.

Event Types
1. **Stock Levels Update:** Notifications of a stock level update in case of any problem, receipt, or transfer of stock.
2. Stock movement, on the other hand, is the movement of items inside the warehouse, meaning the moving of goods from one bin to another.
3. **Stock Adjustments:** For manual or automated changes in stock level due to an audit of inventory, damage, or expiration.

Format and Protocol
JSON/XML or Database Triggers- may be SQL Events on data changes. Protocol: RESTful APIs, CDC events from a Database, and file-based updates like CSV [8].

Challenges
1. Real-time synchronization with inventory systems is critical for maintaining accurate stock levels, requiring the adapter to support real-time processing.
2. Handling race conditions and ensuring event ordering is crucial, especially when multiple events arrive simultaneously from different sources.

Shipping and Logistics Systems
Information about merchandise transportation, shipment tracking,

delivery confirmation, or exception events that must be processed within the WMS to notify customers and in support of resultant activities, such as the initiation of replenishment or return processes.

Event Types

1. **Shipment Tracking Updates:** Real-time updates on the location and status of shipments.
2. **Delivery Confirmation:** Notification that a shipment has been successfully delivered to the customer or warehouse.
3. **Exception Handling:** Information about delays, damage, or other issues encountered during transit.

Format and Protocol

JSON, XML, or EDI (Electronic Data Interchange) formats.
Protocol: RESTful APIs, EDI systems, or file-based integrations (e.g., CSV).

Challenges

1. EDI format parsing can be complex and requires specialized libraries.
2. Exception handling and reconciliation may involve complex workflows, requiring state management within the adapter.

Internal Systems

Events might also be created from internal systems and subsystems in the warehouse, such as conveyor systems, robotic pickers, or storage systems. Normally, events serve to check the condition of the operational equipment of the warehouse and optimization of workflows.

Event Types

1. Conveyor System Events: Events related to the motion of items through conveyor belts, jamming, or system breakdown.
2. 2. Robotic Picker Events: Updates on the status of automated pickers, such as picking success or failure.
3. 3. Events of Storage Systems: The notification of items stored or retrieved or relocated within the warehouse.

Format and Protocol

JSON, XML, or some Proprietary format used by the devices themselves.
Protocol: Message Queues, RESTful APIs, or WebSockets.

Challenges

Integration into proprietary systems requires specific adapters for each subsystem. Real-time processing will prevent any delay in the working of the warehouses.

Benefits of using the Adapter Pattern

Among the many benefits, the Adapter design pattern contributes to a great extent to the tasks of implementing its concept into the architecture of the WMS to cope with multiple heterogeneous data sources. The following sections will elaborate on these advantages in detail and identify the ways they help in creating a robust and maintainable system.

Improved Modularity and Maintainability

Of all the advantages inherent in the use of the Adapter pattern, probably the biggest is related to good separation of concerns [9]. Each adapter knows how to convert data derived from a source in one format into some form of a standard format that the WMS can then use, decoupling core event-handling logic from source-specific implementations. This would result in the huge advantage of modular development whereby, independent of the main workflow of event handling, either an update, addition, or replacement can take place without affecting it.

For instance, any change in the API of the supplying partner will result in changes being done to only the relevant adapter, whereas the core WMS is kept intact. Thus, it would be easily maintainable with minimum possibility for bugs during updating.

Reduced Code Duplication

Without the Adapter pattern, a developer commonly creates his/her event handlers to every new source of data. Thus, such redundancy can grow a lot in the codebase, causing extra effort for maintenance and error-prone code. An Adapter pattern lessens this problem since there is a single interface which each of the adapters will implement. In cases like these, event handling logic can be migrated into the core of the WMS, thereby sharing between distinct event-handling activities-avoiding duplication and hence promoting the reusability of code.

That may be the case, where the core event handler would implement some common application-wide logging logics for WMS applications, such as auditing, while each of the adapters then must perform just the source-specific transformations. There is no code duplication present in such a case; hence the DRY principle remains met, cleaning up the codebase, thereby making it easier to be maintained [10].

Extendable to Handle Future Requirements

It is easy to extend for new use cases that the WMS will need to support either for new event types or data formats using the Adapter pattern. New business needs such as supporting new event attributes may lead to the avoidance of effortful event transformations in the near future that will be carried out by changing/turning the existing adapters without touching the core system. A more specific example would be that in the near future, WMS will have to process incoming updates on inventory data, including additional metadata such as perishables with expiration dates. InventoryAdapter will then parse this metadata from the message and append it to the standardized event format. Thus, should business requirements change, so also does the system, all while keeping its very core architecture stable.

Support for Multiple Protocols and Formats

This adapter design pattern would find immediate practical application in WMS, for instance, since the latter needs to support other systems that may be based on other protocols or message formats. Perhaps one uses RESTful interfaces for event publishing, another uses messaging queues, and yet another uses file-based integration with CSV or XML [11]. There is a common outcome of supporting different data formats and ways of communication: the embedding of the peculiar logic within the WMS itself for each protocol.

The Adapter design pattern encapsulated each of the protocol-specific logics within their respective adapter. All that the WMS knows is how to invoke the processEvent() method on the adapters, to abstract the details of protocol handling. It would now be much easier to integrate multi-protocol capability: it would support several different communication mechanisms and data formats uniformly at the WMS level.

Support for Legacy Systems

Quite often, WMS needs to be integrated with legacy systems that do not consider modern event-based architecture. These most probably will make available the data in some sort of obsolete

format or using obsolete communication mechanisms. This would be highly expensive and dangerous to make changes directly in the WMS to cater to those.

The Adapter pattern will allow this through non-intrusive integration of the legacy systems by the creation of adapters, translating the legacy formats and protocols into standardized formats expected by the WMS. This can be further extended to allow existing coexistence between these legacy systems and modern systems for easier migration and reduction of technical debt from legacy integrations.

Case Study: Implementing Adapters for Inventory and Order Systems

Let's consider a case where a WMS may want to receive changes of inventory from a legacy database hosting inventories and changes of orders from a modern e-commerce platform:

Inventory Adapter

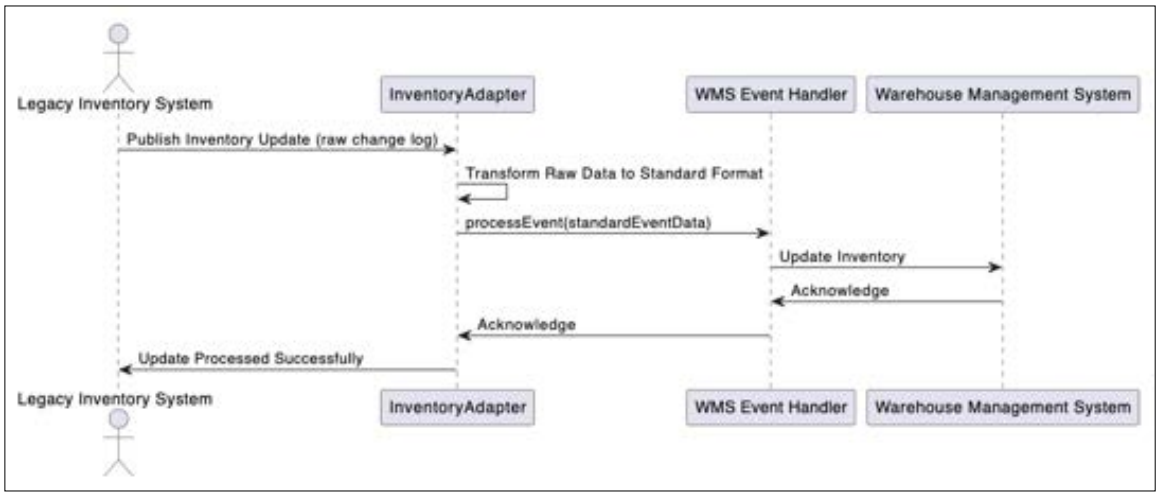

Figure 4: Inventory Event Adapter

This adapter takes all records in the changelog of the database-inventory creations, deletions, and updates-and normalizes them to a standard event structure. The adapter converts the different legacy formats and enriches the event data with the additional information on product categories and supplier details.

Order Adapter

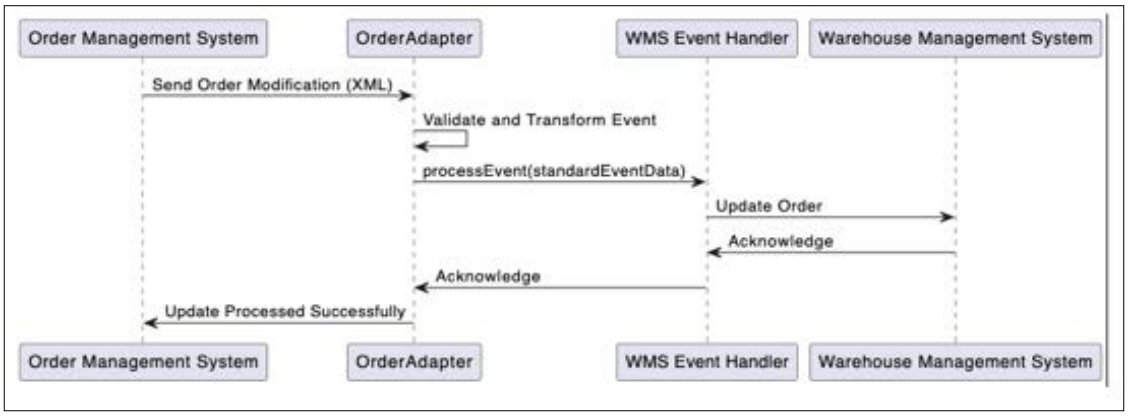

Figure 5: Order Event Adapter

will connect to the RESTful API of the given e-commerce platform to download order events and transform them to the unified event structure. It probably will be interested in protocol-specific stuff like authentication, pagination, and rate limits too, so that the WMS core will get only well-defined order events. These adapters will enable the WMS to process events from any of these two providers without embedding database or API-specific logic into its core modules. It significantly results in a much cleaner, easier-to-maintain system that will adapt quite easily to any changes in the future.

Conclusion

The Adapter design pattern provides the many advantages in event handling in the context of a WMS. It decouples the WMS core from any dependencies that may change or extend its functionality, thus allowing extensibility, scalability, and maintainability; it allows easy integration of new data sources without code duplication; it provides more ease in unit testing and debugging processes. After noting the case study and advantages put forth, one will understand just how much of a utility the Adapter pattern is in the management of complexity within the event handling scenarios of WMS.

References

1. Alves V, Borba P (2001) Distributed adapters pattern: A design pattern for object-oriented distributed applications. in First Latin American Conference on Pattern Languages of Programming—SugarLoafPLoP, Rio de Janeiro, Brazil.
2. Mao J, Xing H, Zhang H (2018) Design of intelligent warehouse management system," Wireless Personal Communications 102: 1355-1367.
3. Johnson RE, Foote B (1988) Designing reusable classes. Journal of Object-Oriented Programming 2: 22-35.
4. Kramer C, Prechelt L (1996) Design recovery by automated search for structural design patterns in object-oriented software. in Proceedings of WCRE'96: 4th Working Conference on Reverse Engineering IEEE.
5. Tretola G, Zimeo E (2007) Extending web services semantics to support asynchronous invocations and continuation. in IEEE International Conference on Web Services (ICWS 2007) IEEE.
6. Apache Kafka Documentation. https://kafka.apache.org/.
7. Atieh AM, Hazem Kaylani, Yousef Al-abdallat, Abeer Qaderi, Luma Ghoul, et al. (2016) "Performance improvement of inventory management system processes by an automated warehouse management system. Procedia Cirp 41: 568-572.
8. Shi J, YuBin Bao, FangLing Leng, Ge Yu (2008) Study on log-based change data capture and handling mechanism in real-time data warehouse. in 2008 International Conference on Computer Science and Software Engineering, vol. 4. IEEE.
9. De B, Win F, Wouter Joosen, Tine Verhanneman (2002) On the importance of the separation-of-concerns principle in secure software engineering. in Workshop on the Application of Engineering Principles to System Security Design.
10. What is DRY Development?. [Online]. Available: https://www.digitalocean.com/community/tutorials/what-is-dry-development.
11. Richardson L, Ruby S (2008) RESTful Web Services. O'Reilly Media, Inc.