Behavior Modeling for Decision Support within a Spare Parts Supply Chain

by

Jean Olivier Sens Luna

Supervisors:

Prof. Dr. Ralf Möller (TUHH)

Dipl.-Ing. Helge Fromm (EADS)

Institute for Software, Technology and Systems

Technical University Hamburg–Harburg

Submitted in partial fulfillment of the requirements for the degree

Master of Science in Information and Media Technologies

Hamburg, March 2009

Confidentiality Agreement

The present Masters Thesis contains confidential information which is intellectual property of EADS Deutschland GmbH. For protection of this information, this Masters Thesis is subject to a publication restriction.

Beyond EADS Deutschland GmbH, only the Masters Thesis supervisors from the Hamburg University of Technology (TUHH) are entitled to use the information contained in this work.

The publication, reproduction or disclosure of the present Masters Thesis or the information it contains, it is only possible by written permission issued by EADS, 81663 München.

EADS Innovation Works
Nesspriel 1, 21129 Hamburg
Germany

Declaration

I solemnly declare that I have written this work independently, and that I did not make use of any aids other than those acknowledged in this document. Neither this thesis, nor any similar work, have previously been submitted to any examination board.

Hamburg, 12th March 2009
Jean Olivier Sens Luna

Acknowledgments

This report describes the thesis project that I developed from September 2008 to March 2009 as part of my studies at the Masters in Information and Media Technologies in the Hamburg University of Technology (TUHH). The project was conducted in the EADS Innovation Works office in Hamburg.

I would like to thank my parents for their inconditional support and the education they have given me, because this has been the basis for my ambition of pursuing graduate studies in Germany.

I am also grateful for the opportunity I was given in EADS and the support of my supervisor Dipl.-Ing. Helge Fromm during the project. I would also like to thank my working colleagues for the valuable advice they provided me.

To Prof. Dr. Ralf Möller for guiding me on the topic and for the helpful insights from our meetings.

Last but not least I would like to thank all the good friends that I had during these years in Hamburg for their support. I would like to give a special mention to Krystyna for her patience, comprehension, and constant motivation during this journey.

Abstract

A Spares Order Desk is a working division within the aviation industry that must provide the supply of spare parts in the appropriate quality, performance, time and costs. One key activity is the planning of delivery routes for the efficient shipment of spares to the customers. Due to emergency situations such as aircraft-on-ground, this activity must be realized in the shortest possible time. The thesis proposes an initial route planning framework that finds the best possible route among a network of transportation means and specific constraints. Furthermore, an optimization mechanism centered on case-based reasoning is proposed, enabling the system to learn from previous addressed routing situations, thus suggesting solutions in a shorter time based on analogies made over the recorded situational behavior.

Index

List of Figures

List of Tables

Preface

Chapter 1: Introduction
This chapter explains the source of the project's target problematic: best routing assessment within a spare parts supply chain. The project motivation and scope are also described, including the proposed solution approaches.

Chapter 2: Route Finding with the Shortest Path Problem
This chapter details the spare parts supply-chain process in the aviation industry. An initial solution approach based on the shortest path problem is described and its implementation is also conducted.

Chapter 3: Routing with a Case-Based Reasoning approach
This chapter describes a solution approach using case-based reasoning. This technique present means of solving new problems based on similar situations already solved, providing a way for modeling the behavior of a system. The integration of approaches is proposed as the best way of addressing the project problematic.

Chapter 4: Web-Based Integrated Solution
This chapter describes the implementation of a multi-tiered Web-application that integrates the solution approaches described in chapter 3 and chapter 4.

Chapter 5: Results
This chapter presents a brief evaluation of the project, describes an experiment carried out for testing the proposed solution and discusses the results obtained.

Chapter 6: Conclusions
This chapter presents a summary of the achieved goals, discusses the identified project shortcomings and proposes future possible work to be done for this problem.

Abbreviations

AOG	Aircraft On Ground
ATA	Air Transport Association
CBR	Case-Based Reasoning
CID	Customer-ID
CPO	Customer Purchase Order
EADS	European Aeronautic Defence and Space Company
JSP	Java Server Pages
NNM	Nearest-Neighbor Matching
PNR	Part Number
PRY	Priority
RDBMS	Relational Database Management System
RTN	Routine Order
SPEC2000	E-Business Specification for Materials Management
SQL	Structured Query Language

ADT	Arrival Delay Time
AMS	Aggregate Match Score
DDT	Departure Delay Time
MWT	Maximum Wait Time
RDT	Real Departure Time
RAT	Real Arrival Time
RQ	Route Quality
TET	Transport Expected Time
TST	Total Shipping Time
TT	Transport Time
TTD	Time to Depart
TTT	Total Transport Time

Chapter 1

Introduction

1.1 Background Information

The project fundamentals lie within the supply chain management of aircraft spare parts. A spare part is a term used to indicate parts of a mechanical item that are used for reparation purposes. Companies conduct a spare parts management process to ensure that the right spare part is at the right place and at the right time for repairing a broken counterpart. Spare parts management plays a critical role in the effective operation of industrial supply chain systems.

A Spares Order Desk is a dedicated aviation manned logistics service that has to provide the spare parts for an operating aircraft fleet in the appropriate quality, performance, time and costs. Its main functions are the alignment of provisioning strategies to spare parts (e.g. on-demand or consumption based allocation and sourcing, central or decentralized storage, etc.) and the operational spares decision support (e.g. the assignment of spares to orders and the restock of the warehouses). One key activity is the efficient determination of spares shipping routes that satisfy the customer requests.

Nearly every aircraft manufacturer and bigger airline maintain a Spares Order Desk to service calls from customers (mainly airlines, military and governmental divisions) searching for critical spare parts during an AOG situation.

AOG (Aircraft on Ground) is a term in aviation maintenance indicating that a problem is serious enough to prevent an aircraft from flying. Generally there is a rush to acquire the parts to put the aircraft back into service and prevent further delays or cancellations of the planned itinerary.

In order to mitigate an AOG status, if the materials required are not on hand, spare parts and personnel must be immediately taken to the location of the grounded aircraft. Conservative estimates calculate that a one-to-two-hour AOG situation costs an airline up to $10,000 in downtime. However, actual costs of such delays can be as high as $150,000 per hour depending on the airplane model or the airline [5]. Due to this exorbitant costs, time becomes the most important asset when determining a delivery route, meaning that the shortest-possible shipping time is the most important goal in this kind of emergencies.

Moreover, a Spares Order Desk also supplies parts to customers in situations such as initial provisioning of spares and the continuous supply of parts that are needed during normal maintenance and servicing. Given that an airplane comprises an approximate of 4 million parts, this is also not an easy task.

Initially, a customer contacts the Order Desk for determining the details of the problematic. For such, a detailed formal request must be filled out to comprise a Customer Purchase Order (CPO). These orders can have either a Routine priority (RTN) or an AOG priority. An order with a RTN priority is a customers' planned need of spares which is usually solved within a controlled environment. These requests are commonly planned and forecasted in such a way that they do not represent a high risk for the company, neither for the customer.

For instance, let us pretend that Lufthansa Technik is a customer that knows its private jets fleet consumes a rate of 30 jet engine replacements per year. The customer specifies preferences such as the amount of parts to be sent at determined periods of time to predefined delivery locations. Since the Order Desk knows these preferences in advance, it plans scheduled supply chains considering a myriad of constraints (i.e. parts stock availability, current flight schedules, existent transport operators) to come up with a spares delivery service level that satisfies the customer's demands.

Nevertheless, this route planning process becomes particularly complicated when emergency situations like AOG occur. In such cases, RTN routes usually do not offer a suitable solution due to the unplanned nature of the event and the high variability of the routing constraints. Currently, the Order Desk does not have an efficient solution model that is able to deal with the common AOG situation, consuming high amounts of time and human resources by analyzing each situation independently. This opportunity area is the basis for the Masters Thesis motivation.

1.2 Project Motivation

The project motivation arises from the decision-making difficulties found when an AOG situation is analyzed by a Spares Order Desk for determination of a time-efficient spares delivery route. The main goal is to provide automatic means of compiling AOG situations, based on a set of constraints, for proposing a best-possible shipping route to deliver the urgently needed spares. This would be the first part of the project.

For this part, a brief research in spares parts management procedures was conducted to understand and identify the relevant factors that influence the decision-making process for best routing [12] [13] [16].

However, to present an algorithm for the computational implementation of a solution approach, a deeper research in graph theory was realized to understand and reuse the principles for solving the *shortest path problem* [6] [7] [9] [26] [27]. The shortest path problem is a method for finding a path between two nodes such that the sum of the weights of the edges that connect them is minimized. A typical implementation of this method are automated routing advisors where the nodes represent locations and the edges represent roads or routes weighted by attributes like time, distance, or any other measurable factor.

The second part of the project has to do with finding an alternative solution approach that complements the shortest path problem by modeling the behavior of the Spares Order Desk. This approach should find an efficient way to reuse experience from previously solved AOG situations. This experience should help to propose solution suggestions for new incoming AOG situations.

For this part, an extensive research on *Case-based Reasoning* (CBR) was conducted to understand and reuse this technique for modeling the behavior of the Order Desk [1] [4] [14] [17] [18] [25]. CBR is a problem solving mechanism that enables flexible reuse of experience. It provides methods that allow a computer system to retrieve and adapt solutions to previous problems for solving new similar problems.

Existing information on previously used AOG shipping routes and attributes in the spares orders must be analyzed to find pattern features among them and build up complete cases. Then, cases could be compared to each other by means of discovering similarities between them, enabling the system to identify which case fits better for a new incoming situation.

Efficiently implemented, this problem-solving mechanism could greatly optimize the decision-making process during AOG situations. CBR is not only a method for computer reasoning, but also is related to common everyday human problem solving based on past cases personally experienced. It has been proven that CBR is a powerful method for problem domains like diagnosis, planning, and decision-support.

1.3 Project Scope

The project aims to complete the four main activities depicted in Figure 1.1.

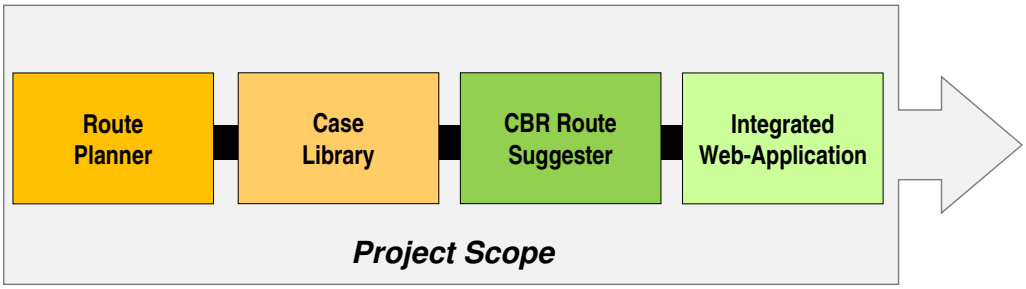

Figure 1.1: Project Scope Concepts

Each activity follows a rationale for its consideration in the project:

- Implement a *Route Planner* for incoming Orders taking in consideration current updated information from spare parts, transportation schedules, available locations, etc. This approach should give a basis framework for best-routing within AOG situations.
- Accumulate the solutions of the Route Planner with their respective Orders in a *Case Library* for future retrieval and reuse. This will be the repository that organizes the routing plans for Orders already processed. This is the Spares Order Desk's captured domain-knowledge.
- Propose an alternative solution approach that optimizes the Planner's processing times by means of analogy. A *CBR Route Suggester* is to be implemented in order to solve new situations based on solutions found for old situations.

- Develop a *Web-Application* that integrates the Route Planner and the Route Suggester under a common interface, enabling the Spares Order Desk with the possibility of inputting incoming orders for decision-support.

1.4 Specific Approaches

The following are the general approaches to achieve the aforementioned activities:

- For the Route Planner: revision of the shortest-path problem and adaptation of the Dijkstra's algorithm implementation.
- For the Case Library: development of a relational database that holds the system's data structures and information for route calculation, as well as the set of processed orders and their solutions representing the Order Desk knowledge-base.
- For CBR Route Suggester: implement a case matching algorithm to compare attributes of new orders with those stored in the case library; implement a case ranking mechanism to select the best-possible case among the matches by means of similarity techniques; implement a case adaptation process that updates old solutions with current routing information.
- For the Web-Application: development of a Web-oriented multi-tiered system architecture, having the relational database as the data tier, the implementations of the Planner and Suggester as the logic tier, and dynamic Web pages as the presentation tier.

Chapter 2

Route Finding with the Shortest Path Problem

2.1 Background Information

2.1.1 Common Spares Delivery Process

The Common Spares Delivery Process is the procedure that a Spares Order Desk follows as a standardized way of processing orders. It is usually organized according to the E-Business Specification for Materials Management (SPEC2000) issued by the Air Transport Association (ATA) [3]. The top-level simplified process is depicted in Figure 2.1.

Send CPO is the first step and it is done by the customer, directed to the Spares Order Desk. The CPO is usually sent via an e-commerce platform connecting both parties and must be formatted as a SPEC2000 message. In the second step, the Order Desk collects and places the CPO in a SAP-based Materials Management System, and sends an acknowledgement receipt to the customer. This is not a contractual affirmation, but means that the CPO is already booked.

During the Commercial Technical Check, the order requests are verified for integrity, the amount of parts requested is checked in the dispatch warehouse, and a delivery route is determined unless one was previously specified. This is the part of the process where the information bottleneck occurs, and a decision-support tool is needed for suggesting a best-possible spares shipping route. After this analysis is done and a route is decided, the CPO details are changed to reflect the decision taken and the detailed spares shipping plan.

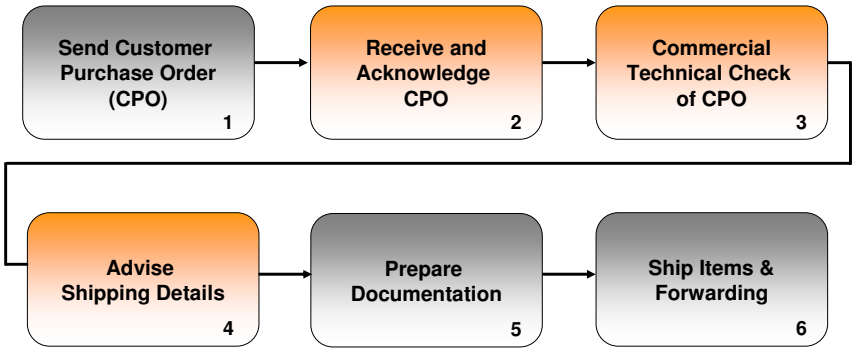

Figure 2.1: Top-Level Simplified Spares Delivery Process

The updated CPO is communicated to the customer, providing the date when the spares are ready for dispatch and the expected final delivery date. Finally, all the official documentation for this order is collected and the spares are forwarded by the chosen transport operators to the defined delivery location.

2.1.2 Shipping Policies

The following are the main policies that constrain the decisioning process:

a) Consolidated Shipments

The standard process of shipping an order is done by consolidated shipments; this means that whenever a set of spares is requested, the full amount of items solicited must be dispatched from the same warehouse. Let us pretend a customer orders 20 pieces of a standard jet accumulator and sets Singapore as its desired dispatch warehouse; if this warehouse have only 15 pieces available, then it is decided that the items will be shipped from a warehouse that completely suffices the order, for instance Hamburg. The reason for this policy is that, in emergency situations, segmented shipments add a higher level of logistical costs and complexity, thus compromising even more the AOG mitigation process.

b) Forwarding Constraints

Due to the unexpected nature of an AOG situation, time is the primary delivery constraint, but also the following policies are to be considered for decisioning:

- The Spares Order Desk designates the set of possible dispatch warehouses depending on the stock availability of the part requested.

- The main criteria determining the best route to follow is the *total shipping time* which is determined by the aerial proximity between locations, expected delay times, and the flight schedule availability.
- Whenever two or more possible routes satisfy the shipment at its best, the differentiation criteria is the *total shipping cost*. However, if this cost is considerably higher than the cost of the AOG situation for the time needed to deliver the spares to mitigate it, then this route should not be considered.
- The spare parts attributes such as dimensions (height, weight, length) and type of part (hazardous, costly piece) determine the possible set of transport operators for forwarding the spares (e.g. not all operators ship hazardous materials).
- Some customers predefine for a specific combination of spare part and delivery location a predefined shipping route with specific transport operators. When these shipping instructions are defined under an agreement even though the order is Routine (RTN) it is treated with AOG priority. Some of these agreements establishes a maximum delivery time, therefore if it is foreseen that this time will not be satisfied, the shipment is dropped and the part is delivered directly to the customer from the spares supplier's warehouse.

2.2 Domain Representation

The spares routing problem involves a set of identified entities that must be represented and organized for understanding the underlying domain information. The complete set of identified system entities is depicted in Figure 2.2.

The main input criteria is an *Order* containing a requested spare part, the customer that requires it, the desired warehouse to ship it from, and the final delivery destination.

The main output criteria is a *Route* and it is integrated by a sequence of scheduled transportation legs which happen in a specific date and time. Each leg has a departure location, an arrival location, and an operator that executes the actual transportation. Locations represent places in the real world that are relevant for the spares forwarding process.

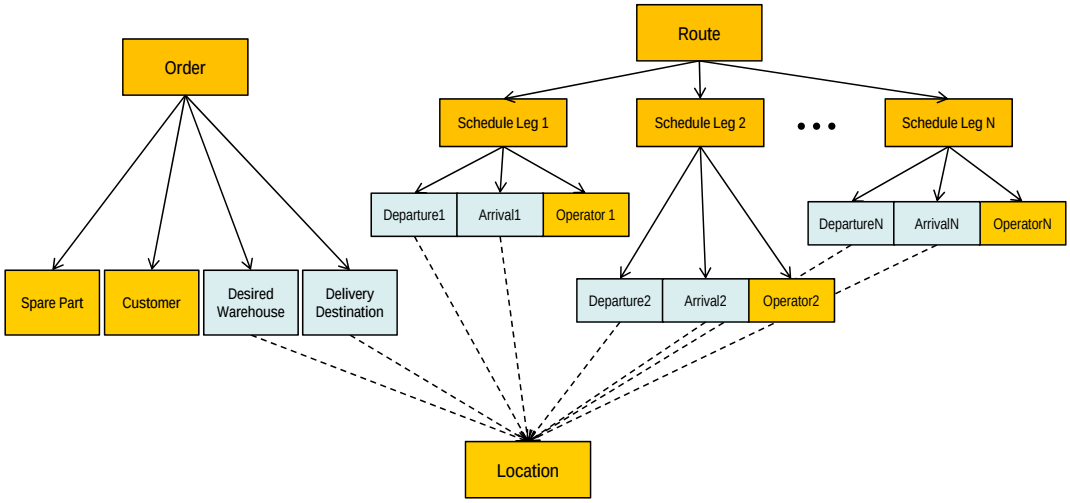

Figure 2.2: Relevant Identified System Entities

The types of identified locations for the problem domain are: the desired warehouse to ship the order from; the delivery destination where spares are to be sent; the departure location of a transport leg; the arrival location of a transport leg.

Additional to this conceptual representation, a database model is necessary for establishing the basis for a practical solution implementation and for determining the necessary information to be considered for the project. In most of the cases, business processes information is stored in relational databases as a centralized way of understanding and keeping information available for a purpose. By providing such a database, the data structure is preserved and the integration with real production databases is a straightforward process. The domain data representation for the project is shown in Figure 2.3 in the form of a relational database diagram.

From this model more detailed information can be derived for implementation purposes. For instance, the table *Location* consolidates all the places that are relevant for the route planning process, such as warehouses, airports, ground transportation warehouses, customer's workshops, etc. A *Route* will be defined by a sequenced order of locations to be visited in order to reach a final destination.

The geographical position of these locations is irrelevant for the project. The *transport expected time* is the criterion to determine how *close* a location is to other locations. This time can be the aerial or ground proximity between these locations.

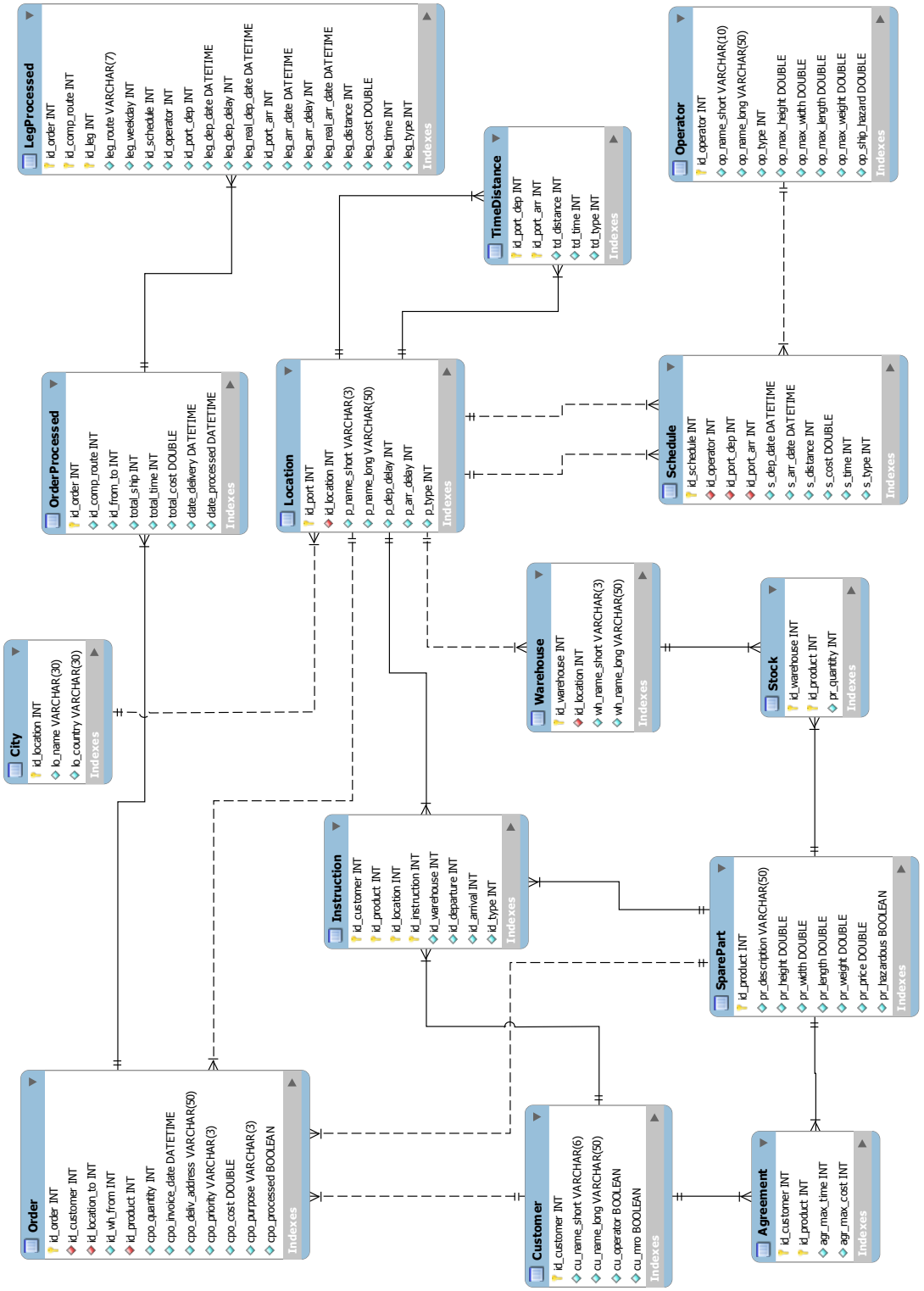

Figure 2.3: Domain's Relational Database Model

The table *TimeDistance* defines all the values for the *transport expected time* between all locations. Aerial transport time expectations are established by the standard flight time between airports. Ground transportation time expectations are established by historical experience.

However, the basis for determining that between two locations exists a transportation leg is the *Schedule*. In this table, the planned transportation activities are defined by departure and arrival locations, and specific departure and arrival dates and times. Each schedule also provides estimated distance in kilometers, expected arrival and delay times, and total cost. These estimations are given by the Spares Order Desk from recorded experience. If a transport operator offers transportation between two locations, for instance every hour, there will be a record for each single transport event.

The *SparePart* table defines the product attributes: length, weight, height, and hazardousness. The *Operator* table defines the set of existing forwarders and their spares measurement limits for transportation purposes. This information is used to constrain the available schedule legs by allowed operators given the characteristics of the product to be dispatched.

The *Customer* table contains the collection of customers of the Order Desk. It is related to the *Instruction* and *Agreement* tables. The first defines the combinations of spare part and delivery locations that are to be considered within an agreement, and the respective sequence of legs to be followed to comply with it. The second defines the maximum time and maximum cost that each agreement defines, this is control information to derive if proposed routes meet agreements properly.

The *Warehouse* table defines the existing warehouses that dispatch spare parts to customers. The *Stock* table defines the current amount of spares available in each warehouse.

Finally, the *Order* table defines all the attributes for a Customer Purchase Order. Respectively, *OrderProcessed* and *LegProcessed* collect the historical information of processed orders and the routes they followed to deliver their requested spares. This information is will be the basis for implementing the case-based reasoning approach mentioned in the project scope.

In order to build a routes network for path finding, database tables *TimeDistance* and *Location* can be crossed to obtain the necessary input information. Table 2.1 is an excerpt of the obtained data matrix showing how much time is expected to traverse a route between locations.

HAM	IAD	DXB	SIN	PEK	CDG	WPL	MEX	WVM	SYD	WQC	—
0	∞	440	800	∞	120	560	∞	∞	∞	∞	HAM
∞	0	∞	∞	980	540	∞	470	∞	∞	∞	IAD
440	∞	0	∞	∞	470	∞	∞	∞	1010	∞	DXB
800	∞	∞	0	∞	∞	∞	∞	∞	600	∞	SIN
∞	980	∞	∞	0	∞	∞	∞	∞	860	∞	PEK
120	540	470	∞	∞	0	60	∞	∞	∞	∞	CDG
560	∞	∞	∞	∞	60	0	∞	∞	∞	∞	WPL
∞	470	∞	∞	∞	∞	∞	0	240	∞	∞	MEX
∞	∞	∞	∞	∞	∞	∞	240	0	∞	∞	WVM
∞	∞	1010	600	860	∞	∞	∞	∞	0	330	SYD
∞	∞	∞	∞	∞	∞	∞	∞	∞	330	0	WQC

Table 2.1: Excerpt matrix of expected transport times

The header identifiers are standard abbreviations for airports and aircraft workshops (i.e. HAM means Hamburg Fuhlsbüttel Airport, WQC means Workshop Qantas Canberra, etc.), the numbers contained are transportation times in minutes, and the ∞ (infinity symbol) means that there is no transport link serving this route. Figure 2.4 is a representation of a *Routes Network Graph* built from the sample data on the previous table.

Figure 2.4: Routes Network Graph superimposed on a world map

2.3 The Route Planner

Once the information is structured, it is possible to propose an initial solution approach for the route determination process. Therefore, a *Route Planner* is to be defined and implemented for efficiently finding the shortest possible path given a routes network and a departure and destination location. This implementation is based on the shortest path problem and reuses the principles of the Dijkstra's algorithm for its solution.

2.3.1 Dijkstra's Shortest-Path Algorithm

A shortest path algorithm is a technique that tries to find a path between two nodes such that the sum of the weights of its edges is minimized. The term *shortest* is defined as the minimum possible sum of the weights of the edges connecting the nodes that the path traverses, also called the *length of the path*. Weights can be any kind of applicable measurement: distance, time, cost, etc.

Dijkstra's algorithm is a very popular and efficient algorithm that solves the shortest path problem. It has been successfully implemented in many route planning problems [7] [15] [19] and therefore it was chosen as the solution mechanism for the problem domain. Dijkstra's algorithm is explained in the following five steps:

1. **Assumptions:** To compute the shortest path from a given start node s to a given end node z of a given graph G, all the paths from s to all other nodes must be traversed to assure that the selected-to-be shortest path is actually the shortest. For accumulating the weights of the traversed paths, an array D is maintained and indexed by the set of existing nodes. When the algorithm finish its execution, $D[z]$ will contain the total weight of the shortest path from s to z.

2. **Initialization:** Before starting the algorithm, it is mandatory to set $D[s] = 0$ (meaning that the distance of the start node to itself is 0) and $D[z] = \infty$ for the rest of the nodes (meaning that the distance to all other nodes is still unknown and its considered as infinite); this unconfirmed distance value is called *overestimate*. The algorithm will decrease this overestimate until is no possible anymore and then will terminate.

3. **Validation:** The core validation of this algorithm is to find edges which weights can decrease the path overestimate. Let us suppose there is a

node b, which has an edge to z with a weight value of $weight[b][z]$. If the sum of the overestimate $D[b]$ and the given $weight[b][z]$ is smaller than the overestimate $D[z]$, then a shortcut was found. This means that the total length of the path $s \rightarrow b \rightarrow z$ is smaller than the distance overestimate from s to z, namely $D[z]$. Therefore, the overestimate $D[z]$ can be replaced by the value of following sum: $D[b] + weight[b][z]$, which actually is a better path length estimate. This validation must be systematically done through the entire graph to keep improving the overestimate until the best possible path length is achieved.

4. **Finding minimal estimates:** At each stage of the algorithm, if the distance $D[x]$ of a node x has the minimal value among all values recorded in the array D, then the value in $D[x]$ is considered the minimal possible estimate for the node x. This means that the estimate improvement validation mentioned in step 3 will not find any better path length at this node. If this happens, the node x is then added to a second array P that systematically stores the shortest-path set of nodes.

5. **Finalization:** The algorithm finishes when all the nodes in the graph G have got their minimal possible estimate. A method for extracting the value $D[z]$ and the ordered set of nodes in array P must be implemented for respectively outputting the length of the path until final node z and the nodes that conform the shortest-possible path.

2.3.2 Algorithm Inputs Obtainment

As mentioned above, in order to conduct Dijkstra's algorithm an *input graph* with a defined *start node* and *end node* must be established. The following example describes a route determination situation and the considerations observed to obtain these necessary inputs.

Let us suppose that an aircraft from Qantas Airlines located at a workshop in Canberra cannot operate because eight of its *Fuel Shutoff Valves* are broken and need to be replaced to get the aircraft back to function. Immediately, the customer sends a spares order with AOG priority to the Spares Order Desk. Table 2.2 shows a simplified view of the information contained in such an order. The *Delivery Destination* specified there will represent the algorithm's *end node* for this situation, namely WQC.

Spares Parts Order Sample	
Order-ID	45001
Current Date	Wed 11.02.2009
Current Time	07:00hrs.
Order Priority	AOG
Part-ID	90009 *(Fuel Shutoff Valve)*
Quantity	8 units
Delivery Destination	WQC *(Workshop Qantas Canberra)*
Desired Warehouse	HAM *(Hamburg Warehouse)*
Customer-ID	30003 *(Qantas Airlines)*

Table 2.2: Spares Parts Order Sample

To assess this order, the Order Desk firstly searches the ordered part in all the available warehouses through an automated database procedure that queries the existing spares stock information. The locations of the warehouses found with enough spares availability are going to be the set of possible *start nodes* for the algorithm. Since the algorithm only accepts one start node at a time, the algorithm must be executed as many times as possible *Dispatch Warehouses* are found. For the given example, let us suppose enough spares availability is found in Hamburg (HAM) and Peking (PEK).

Finally, the *input graph* is obtained by determining the current *routes network*. This network is never constant, given that at the reception time of the order some operators might not have scheduled transportation means (edges) connecting locations (nodes). Usually a spares delivery route is integrated by a chronological sequence of transport legs, therefore the routes network must consider transportation schedules for a specific period of time (i.e. transportation departing from the order reception time until a maximum of 72 hours in the future, given the belief that any of the locations served by the Order Desk can be reached within this maximum timeframe). A final constraint must be included to only obtain schedules performed by operators whose product transportation limits allow the measurements of the ordered spares.

The set of nodes integrating the *input graph* will be the locations connected by the obtained scheduled routes. The set of edges' weight values will be the respective transportation times between pairs of locations. These times, as well as the routes depart and arrival times, must be updated to consider experienced delay expectations. This will be discussed in the next section.

2.3.3 Handling of Time Information

The distribution of Time in the implementation follows the sample structure defined in Figure 2.5.

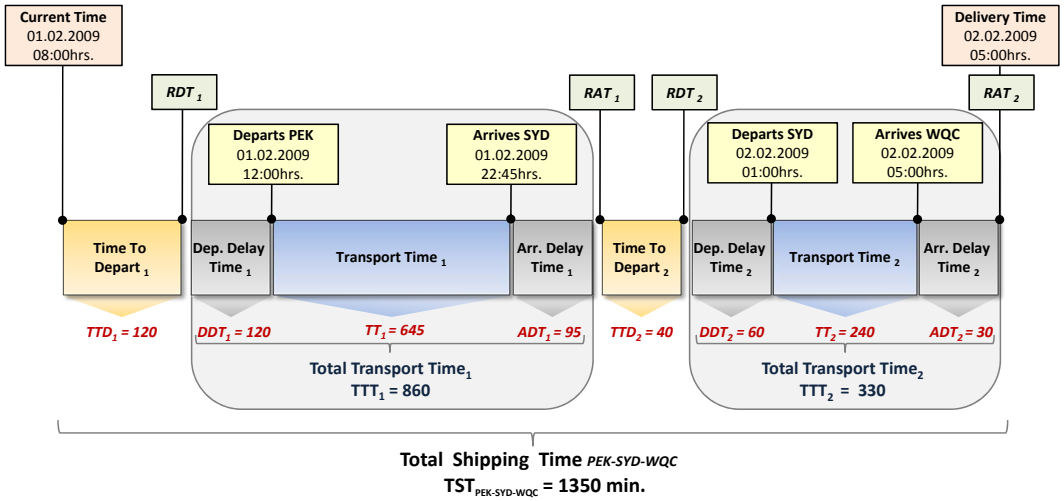

Figure 2.5: Sample Distribution of a Route's Time Information

Every possible route from the given start location to the delivery location is a chronologically ordered set of scheduled transport links. Each of these scheduled legs contains the following time-related information:

- **Transport Time (TT_i):** the amount of time that the actual transportation (aerial or ground) takes for carrying the spares from one location to another. It has a defined *Departure Time* (DT_i) and *Arrival Time* (AT_i). It is considered a unique event by combining location-to-location, time-date, and operator information.

- **Departure Delay Time (DDT_i):** the amount of time that is expected or needed for preparing the departure of the transportation link. This account for boarding time, loading time, customs delays, or the sum of those applying for this link. By subtracting the *Departure Delay Time* from the *Departure Time*, the *Real Departure Time* (RDT_i) is obtained.

$$RDT_i = DT_i - DDT_i \tag{2.1}$$

- **Arrival Delay Time (ADT_i):** the amount of time that is expected or needed for preparing the arrival of the transportation link. This accounts

for unloading time, customs delays, or the sum of both if the two apply. By adding the *Arrival Delay Time* to the *Arrival Time*, the *Real Arrival Time* (RAT_i) is calculated.

$$RAT_i = AT_i + ADT_i \tag{2.2}$$

- **Time to Depart (TTD_i):** for the first leg (2.3) will be used, this is the amount of time allocated between the *Current Time* (CT), which is the timestamp at the moment of the algorithm execution, and the first leg's RDT. For the rest of legs (2.4) will be used, this is the amount of time between the RDT of the current leg and the RAT of the previous leg.

$$TTD_1 = RDT_1 - CT \tag{2.3}$$

$$TTD_n = RAT_{n-1} - RDT_n \tag{2.4}$$

- **Total Transport Time (TTT_i):** the sum of *Transport Time*, *Departure Delay Time*, and *Arrival Delay Time* for each scheduled leg of each route.

$$TTT_i = DDT_i + TT_i + ADT_i \tag{2.5}$$

It is important to mention that all scheduled Departure and Arrival Dates for all transport links are homogenized under the Greenwich mean time. This is to reduce the complexity of dealing with different time zones.

For each possible route a Total Shipping Time (TST) is calculated:

$$TST_{route} = \sum_{i=1}^{n} TTT_i + TTD_i \tag{2.6}$$

Where n is the total amount of transport links for this route. For the Dijkstra implementation this would be the amount of edges traversed between initial node s and final node z.

2.3.4 Algorithm Implementation

The inputs for the Route Planner's algorithm implementation are now set:

1. **A graph G with n nodes connected by weighted edges:**
   (a) Graph: the n obtained locations stored into an array G.
   (b) Edges: the obtained transport schedules pairing the n locations stored into an array $Edge[x][y]$. The initial edge weight value will be the Total Transport Time (i.e. $Edge_{[HAM][DXB]} = TTT_{[HAM][DXB]}$)
2. **A start node s and a given end node z:** following with the example situation described in section 2.3.2, these are the following:
   (a) Route Option 1: $(s = HAM, z = WQC)$
   (b) Route Option 2: $(s = PEK, z = WQC)$

Let us calculate the shortest path for the Route Option 1, supposing that a subset of the data matrix of Table 2.1 is the input graph, represented graphically in Figure 2.6.

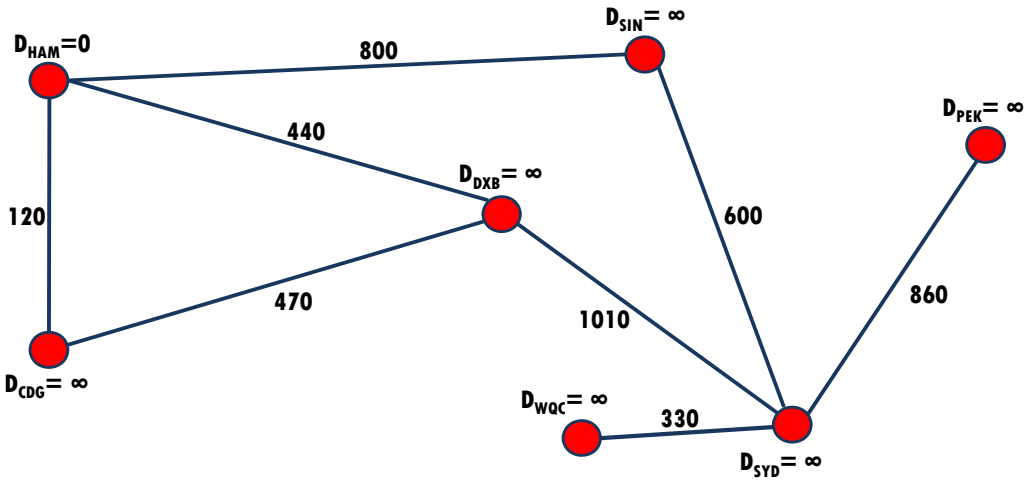

Figure 2.6: Sample Implementation Graph with initial values

To follow the algorithm execution, a control table is provided to show how the data is updated at each stage. The initial values, namely the *Stage 0*, are shown in the following Table 2.3.

—	HAM	CDG	DXB	SIN	SYD	PEK	WQC
length (D)	0	∞	∞	∞	∞	∞	∞
min.estimate	false	false	false	false	false	false	false
via node	—	—	—	—	—	—	—

Table 2.3: Stage 0: Initial set for Calculation

The following are the sample stages taken by the algorithm to reach a solution.

Stage 1: Computing shortest paths from start node $s = HAM$

Node HAM has reached its minimal estimate. Its neighbors have their estimates decreased from ∞ to the value of length D_y to each one of them. The value of D_y will accumulate the length in time to get to location y. To have a more realistic shipping time determination, the value of $Edge[x][y]$ must be updated by adding the *Time To Depart* to the actual expected *Total Transport Time*. Therefore, the realistic value for an edge connecting two locations is determined by: $Edge[x][y] = TTT_{[x][y]} + TTD_{[x][y]}$

- $D_{CDG} = TTT_{[HAM][CDG]} + TTD_{[HAM][CDG]} = (120 + 60) = 180$
- $D_{DXB} = TTT_{[HAM][DXB]} + TTD_{[HAM][DXB]} = (440 + 260) = 700$
- $D_{SIN} = TTT_{[HAM][SIN]} + TTD_{[HAM][SIN]} = (800 + 240) = 1040$

—	HAM	CDG	DXB	SIN	SYD	PEK	WQC
length (D)	0	180	700	1040	∞	∞	∞
min.estimate	*true*	false	false	false	false	false	false
via node	—	*HAM*	*HAM*	*HAM*	—	—	—

Table 2.4: Stage 1: Node HAM has minimal estimate

Stage 2: Computing shortest paths from next node $n = CDG$

Node CDG has reached its minimal estimate and its neighbor HAM is also in its minimal. Its neighbor DXB has its estimate decreased from $D_{DXB} = 700$ to the updated weight via CDG.

- $D_{DXB} = D_{CDG} + Edge_{[CDG][DXB]} = 180 + (470 + 30) = 680$

—	HAM	CDG	DXB	SIN	PEK	SYD	WQC
length (D)	0	180	680	1040	∞	∞	∞
min.estimate	*true*	*true*	false	false	false	false	false
via node	—	*HAM*	*CDG*	*HAM*	—	—	—

Table 2.5: Stage 2: Node CDG has minimal estimate

Stage 3: Computing shortest paths from next node $n = DXB$

Node DXB has reached its minimal estimate and its neighbors HAM and CDG are also in their minimal. Its neighbor SYD has its estimate decreased from ∞ to the updated weight via DXB.

- $D_{SYD} = D_{DXB} + Edge_{[DXB][SYD]} = 680 + (1010 + 60) = 1750$

—	HAM	CDG	DXB	SIN	SYD	PEK	WQC
length (D)	0	180	680	1040	1750	∞	∞
min.estimate	*true*	*true*	*true*	false	false	false	false
via node	—	*HAM*	*CDG*	*HAM*	*DXB*	—	—

Table 2.6: Stage 3: Node DXB has minimal estimate

Stage 4: Computing shortest paths from next node $n = SIN$

Node SIN has reached its minimal estimate and its neighbor HAM is also in its minimal. Its neighbor SYD has its estimate unchanged because the updated weight via SIN is higher than the actual value via DXB.

- $D_{SYD_{[SIN]}} = D_{SIN} + Edge_{[SIN][SYD]} = 1040 + (600 + 160) = 1800$
- $D_{SYD_{[DXB]}} = D_{DXB} + Edge_{[DXB][SYD]} = 680 + (1010 + 60) = 1750$
- $D_{SYD_{[DXB]}} < D_{SYD_{[SIN]}}$

—	HAM	CDG	DXB	SIN	SYD	PEK	WQC
length (D)	0	180	680	1040	1750	∞	∞
min.estimate	*true*	*true*	*true*	*true*	false	false	false
via node	—	*HAM*	*CDG*	*HAM*	*DXB*	—	—

Table 2.7: Stage 4: Node SIN has minimal estimate

Stage 5: Computing shortest paths from next node $n = SYD$

Node SYD has reached its minimal estimate and its neighbors SIN and DXB are already in their minimal. Its neighbors PEK and WQC have their estimates decreased from ∞ to the updated weight via SYD.

- $D_{PEK} = D_{SYD} + Edge_{[SYD][PEK]} = 1750 + (860 + 90) = 2700$
- $D_{WQC} = D_{SYD} + Edge_{[SYD][WQC]} = 1750 + (330 + 30) = 2110$

—	HAM	CDG	DXB	SIN	SYD	PEK	WQC
length (D)	0	180	680	1040	1750	2700	2110
min.estimate	*true*	*true*	*true*	*true*	*true*	false	false
via node	—	*HAM*	*CDG*	*HAM*	*DXB*	*SYD*	*SYD*

Table 2.8: Stage 5: Node SYD has minimal estimate

Last Stages: Computing shortest paths from $n = PEK$ and $n = WQC$

In the last stages it is confirmed that the nodes PEK, SYD, and WQC have already reached their minimal estimates. Finally, when all nodes have reached their minimal estimates the algorithm comes to an end.

Computation Results

The shortest path for this sample implementation is $HAM \rightarrow CDG \rightarrow DXB \rightarrow SYD \rightarrow WQC$. The length of this path is the value of $D_{WQC} = 2110$. A graphical representation is shown in Figure 2.7. For this particular case, it is expected that the ordered spares will be finally delivered in 35 hours and 10 minutes, if departing from the Hamburg Warehouse.

However, during the spares avilability check it is confirmed that there was also enough stock for delivery in the PEK warehouse. This means that the algorithm must be executed again with PEK as start node, in order to confirm which of both path lengths is the shortest.

The solution for the original input order is the route that offers the shortest transportation time. In case two or more routes share a minimal time, the differentiation factor will be the *Route Total Cost*. Nevertheless, all solutions are recorded in the database for possible future use.

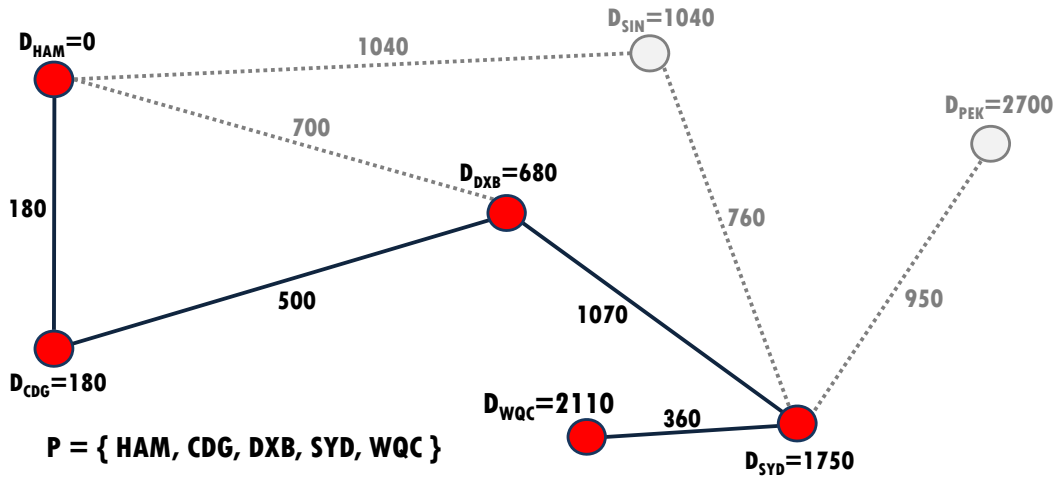

Figure 2.7: Sample Implementation Graph with final values

2.3.5 Algorithm Efficiency

Dijsktra's algorithm traverses graphs doing a *depth first search*, meaning that it expands a start node by pushing all its neighbors nodes in a memory stack and then chooses the next node to expand by taking it from the stack. However, when searching large graphs that cannot be fully contained in memory, this kind of search can suffer from non-termination. The Route Planner's real implementation graph is expected to be large, therefore a search optimization is needed.

For a more efficient search within the Dijkstra's algorithm a priority queue is to be included. A priority queue is an abstract data type in programming languages that sorts elements in a data structure according to a custom comparator, providing constant-time access to the smallest element. This means that each time that a node in the graph is visited, all of its neighboring nodes will be stored in the queue and therefore *prioritized*. Then, the queue will return the node with the highest priority, this is the node with the lowest *total transport time* value. This way, the algorithm avoids to visit all neighbor nodes whenever a new node is reached, optimizing the search and the time for computing routes.

A naive implementation of the priority queue gives a run time complexity $O(N^2)$, where N is the number of nodes in the graph. Implementing the priority queue with a *Fibonacci heap* makes the time complexity $O(E + NlogN)$, where E is the number of edges in the graph [8] [22].

2.4 Conclusions

Given that the Spares Order Desk current procedure for solving AOG situations lacks of an automated solution framework, it is concluded that the Route Planner's implementation of the Dijkstra's algorithm provides the Spares Order Desk with a basic solution model for determining spares delivery routes for incoming AOG situations.

Dijkstra's algorithm is a simple but powerful solution already proven in areas such as networking packet routing and traffic information systems. However, this algorithm may not always be the most efficient solution, especially when the input graph is very dense which is usually the case of real world applications.

The major disadvantage of the algorithm is the fact that it does a blind search, therefore consuming much computational resources. The algorithm can be optimized by including a priority queue during the route search. Nevertheless, even when this optimization considerably reduces the computation time for route determination, another solution approach is to be proposed to take advantage of the experiential knowledge of the Spares Order Desk.

For defining and implementing such a solution, the following chapter addresses the project problematic with a case-based reasoning approach. This technique intends to provide solutions to new problems based on solutions to previous solved cases. Every case and its solution are recorded within a knowledge-base that enables the system to learn from experience. This special kind of problem-solving could dramatically reduce the Order Desk's efforts when determining efficient spares supply routes.

Chapter 3

Routing with a Case-Based Reasoning approach

3.1 Introduction to Case-Based Reasoning

Case-Based Reasoning (CBR) is the process of solving new problems based on the solutions of similar past problems. CBR can also mean adapting old solutions to meet new demands by using old cases to critique new solutions. It is likely to find everyday examples of CBR, for instance a doctor who has a patient with an unusual combination of symptoms, he may remember situations with previous patients where symptoms were similar, thus proposes the old diagnosis as a solution. One advantage of using cases is that they provide context for discussing more abstract issues.

CBR suggests a model of reasoning that incorporates problem solving, understanding, and sustained learning. The latter is achieved when solutions to problems are properly stored in a *case library*. The term *problem solving* is commonly used in a wide sense. It means is not necessarily the finding of a concrete solution to an application problem, but maybe an interpretation of a problem situation, a set of possible solutions, or even justifications to solutions.

There are some premises to be considered when implementing this methodology:

- Reference to previous similar situations is necessary to deal with new problems, meaning that remembering a case is a necessary learning process.
- A reasoner cannot recall a relevant case unless it understands the new situation in it, meaning that specific case representation and matching are also

necessary processes for proper case recognition.

- To compensate the differences between old and new situations a case adaptation process is necessary, because in most of situations old cases are not exactly similar to new ones.
- For supporting the learning process it is necessary to record all solved situations and to have them properly indexed in a case library for future case retrieval.

3.1.1 Definition of a Case

A case is a contextualized piece of knowledge representing an experience that teaches a lesson to achieve the goals of the reasoner. Cases represent specific knowledge tied to specific situations, meaning they provide knowledge at an operational level. Usually, cases record experiences that are different from what is expected. However, not all differences are important to record.

CBR emphasizes on the use of concrete instances for its learning process, because they can provide more guidance in solving a new problem than abstract operators. Cases show application and use of knowledge that abstract operators do not supply.

The premise in CBR is that once a problem has been solved, it is often more efficient to solve the next similar problem by starting from the old solution rather than by rerunning all the reasoning that was necessary the first time. This is precisely the fundamental reason for selecting CBR as an alternative approach for the Spares Order Desk route determination process.

3.1.2 CBR Reasoning Cycle

The following are the primary processes required for Case-Based Reasoning, which are also called the R4 Cycle (depicted in Figure 3.1):

1. **Retrieval:** recall a set of cases similar to the incoming new case.
2. **Reuse:** map retrieved solutions to current problem.
3. **Revision:** adapt a retrieved solution to fit the new situation.
4. **Retain:** record the case and its adapted solution into case library.

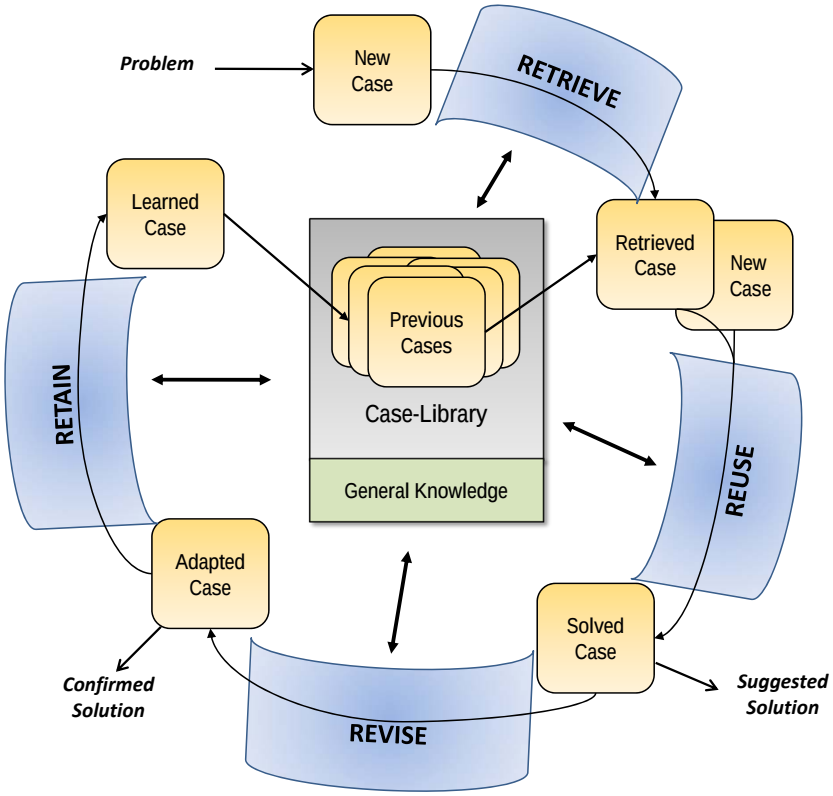

Figure 3.1: Case-Based Reasoning Cycle

An initial description of a problem defines a new case, its features are used to *Retrieve* other cases and its solutions from the case library by means of similarity functions. The retrieved solutions are mapped to the new case through *Reuse*, in order to propose a solution for it. Then, the case and its proposed solution are *Revised* by being tested within the application domain, if solution does not completely fit, it must be properly adapted and tested again. Once the new case and solution are proven, they can be *Retained* for future use by means of recording and indexing them in the case library. The *General Knowledge* is the domain-dependent available knowledge that is not embodied by the cases themselves.

3.2 The CBR Route Suggester

Based on this methodology, the second part of the project is to define and implement a *CBR Route Suggester* for improving the route determination process by means of reusing previously recorded knowledge.

A thoroughly reasearch in this area was conducted in order to understand this methodology and to discover its potential benefits within the problem domain [1] [4] [14] [17] [18] [25]. Moreover, some related work exists that addresses the use of CBR for route planning and for optimization of the shortest path problem [2] [11] [20] [24]. However, like most of the current CBR applications, the goals they achieve are specific to the domains they address, but some useful principles were reused for the design of the Route Suggester proposed in this section.

There are four basic steps to follow when creating a CBR system:

1. The existing data must be examined and relevant features identified to correctly conform the representation of a case.
2. The most relevant case features must be identified for defining the system indexes for efficient case retrieval.
3. A matching mechanism must be defined for mapping new with old cases by means of similarity and ranking functions.
4. Define a case revision process that adapts selected old case's features for fitting new cases.

3.2.1 Case Representation

The process of representing a case involves an analysis of the available domain data to draw a set of features that every case would have in common. Therefore, a *situation description* and a *derived solution* are to be defined for the cases.

a. Situation Description

The situation description encodes the state of the world as reasoning begins and usually represents a problem that needs to be solved. This would roughly be the spares Order entity. A situation description representation of a sample Order is shown in Box 1. It has three major components: *goals* to be achieved in solving the problem, *constraints* on those goals and *features* of the problem situation. Only those fields which would aid the retrieval process are included in the case library. This allows the system to ignore features that are relevant but left implicit because they never vary or because they don't offer any means for indexing or similarity degree matching. The rest of the Order features are maintained in the database system for referential purposes.

```
SITUATION DESCRIPTION:

GOAL: (shortest-time shipping route to WQC)

CONSTRAINTS:
     (set of warehouses with part availability)
     (set of transport operators)
     (set of available transport links)
     (ship-to location)

SITUATION:
     Order-Id: 45001
     Order-Date: 11.02.2009
     Order-Time: 07:00hrs.
     Order-Weekday: WED
     Order-Priority: AOG
     Part-Id: 90009
     Part-Name: Fuel Shutoff Valve
     Part-Quantity: 8 units
     Ship-To-Location: WQC
     Desired-Warehouse: PEK
     Customer-Id: 30003
     Customer-Name: Qantas Airlines
```

Box 1: Sample Situation Description

b. Solution Description

The derived solution incorporates the concepts that achieve the goals set in the situation description, taking into account the specified constraints. In the problem domain, the solution for an incoming spares order is a derived shipping route plan. A sample solution description representation is shown in Box 2. It has three major components: the *solution* itself, the set of *reasoning steps* used to solve the problem and the *justifications* for the decisions taken when solving the problem.

c. Integrated Structure

So far, the representation of a case has been described as a unitary measure, meaning that the integration of an order and its solution represent a complete case for the knowledge library. However, if the case's solution is subdivided into smaller solutions it could be possible to reuse sub-paths of shipping routes for solving new cases.

The subdivision of the situation description provides no benefit, but the subdivision of the solution description has the potential to offer more than one possible solution.

```
SOLUTION DESCRIPTION:

PLAN:
  Leg-1:
   From: PEK
   To: SYD
   DepWeekday: WED
   Departs: 12:00
   ArrWeekday: WED
   Arrives: 22:45
   DDT: 120
   TT:  645
   ADT:  95
   TTT: 860
   Cost: 35.000
   Type: Air
   Operator: Qantas Airlines

  Leg-2:
   From: SYD
   To: WQC
   DepWeekday: THU
   Departs: 01:00
   ArrWeekday: THU
   Arrives: 05:00
   DDT:  60
   TT:  240
   ADT:  30
   TTT: 330
   Cost: 5.000
   Type: Ground
   Operator: TNT Transports

REASONING:
 Step1:
    a-kind-of: case-based inference
    source-case: get-case-from-library(PEK,WQC)
    adaptation: step1-tst[] = get-current-schedule( PEK,SYD,WQC,Weekday[],CurrTime)
         justification: check-wh-stocks( PEK, Part-Id > Part-Quantity )
                        Weekday[] = WED

 Step2:
    a-kind-of: case-based inference
    source-case: get-case-from-library(From[],WQC)
    justification: From[] = get-wh-stocks(*,Part-Id)>Part-Quantity
    adaptation: step2-tst[] = get-current-schedule( From[],*,WQC,Weekday[],CurrTime)
         justification: From[] = get-warehouses( Part-Id > Part-Quantity )
                        Weekday[] = WED,THU

 Step3:
    a-kind-of: case-based inference
    source-case: get-best-similarity-case(step1-tst[], step2-tst[])
    justification: step1-tst[] not null and complete
                   step2-tst[] not null and complete
    adaptation: save-in-library(source-case)


 Step4:
    a-kind-of: case-based inference
    source-case: execute-dijsktra(From[], WQC, get-possible-routes(From[],WQC,CurrTime))
    justification: From[] = get-warehouses( Part-Id > Part-Quantity )
    adaptation: save-in-library(source-case)
```

Box 2: Sample Solution Description

As it was mentioned in the previous chapter, a route plan is defined by the locations it traverses and the scheduled legs that chronologically serve the transportation between these locations, until arrival to the final delivery destination. For Dijkstra's algorithm this is the shortest possible path between a node s to a node z. It is important to mention that a sub-path of a shortest path is itself a shortest path. This means that if the shortest path between node s to node z happens to go through a node x and then node y, then the path from s to z can be split into three sub-paths: $s \rightarrow x$, $x \rightarrow y$, and $y \rightarrow z$. If we were looking for the shortest path between nodes x and z in the same graph, then by definition the shortest path must go through node y.

The CBR Route Suggester reuses this principle by considering *sub-paths* of a solution route as candidate solutions for new situations. This will be further explained in the Case Matching section.

Finally, an Integrated Case Representation is defined to blend both the situation and solution descriptions. Figure 3.2 depicts this representation.

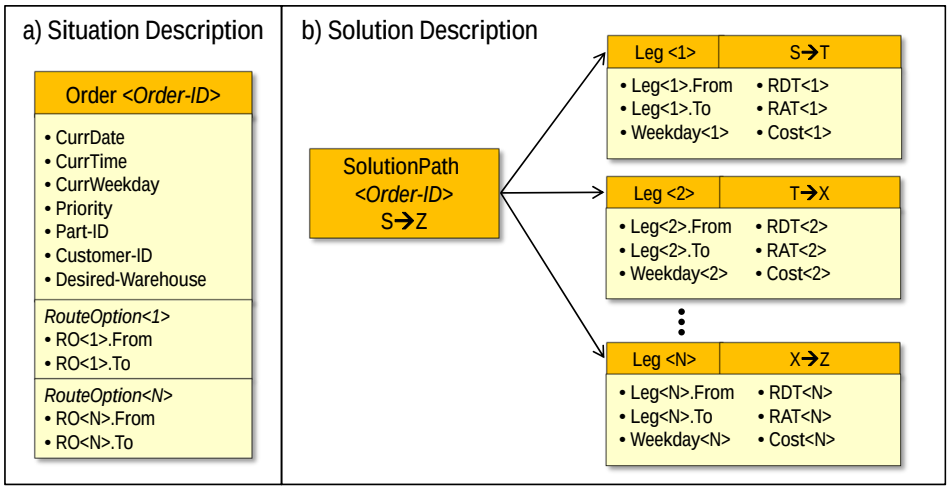

Figure 3.2: Integrated Case Representation

Once a common case representation is defined, the biggest issue in CBR has to be addressed: the retrieval of appropriate cases. The importance of this process is fundamental because if a situation cannot be properly matched to old situations, then the derived solutions will not properly fit the case, meaning that the whole learning process could be compromised.

The *case retrieval* process, no matter the method, requires a combination of search and matching:

- Search: this problem is commonly called the *Indexing Problem* because indexes are the labels assigned to cases that define under what conditions each case can be used to make useful inferences.
- Matching: this problem has to do with finding potentially similar cases from the library and judging their potential usefulness by means of similarity. Then, the retrieved cases can be ranked depending on a defined scale that best meets the reasoner's goal.

3.2.2 Case Indexing

Case indexing involves assigning indexes to cases to facilitate their retrieval. Several guidelines on indexing have been proposed by CBR researchers [25] and even many automated methods have been used to select them. However, despite the success of such methods, Kolodner [14] suggests that indexing is a problem better addressed by humans and therefore for practical applications indexes should be chosen *by hand*.

The following are the general guidelines for choosing indexes proposed by the author:

- Indexes should be predictive.
- Indexes should address the purposes the case will be used for.
- Indexes should be abstract enough to make a case useful in a variety of future situations.
- Indexes should be concrete enough to be easily recognized.

Kolodner also suggests that the usefulness of indexes must be maximized. Therefore, considering that the CBR Route Suggester is a kind of reasoner that use cases to help generate solutions to problems, the most suitable indexes will be combinations of features responsible for choosing a particular solution.

Table 3.1 describes the relevant indexes for the case matching process. It includes the constraints to apply them, their importance order, and basic examples on how they assess the matching.

Case Indexes	
Index Description	**Examples**
1. Location-related: Dispatch-Warehouse (DW), Delivery-Location (DL) 1.1. *DW* is the start location and *DL* is the final location in the solution Path. 1.2. *DW* is any location in the solution Path (except the final location) and *DL* is another location that is chronologically after *DW* in the solution path. (i.e. the start and end nodes from a sub-path of the solution path) 1.3. If the new case contains more than one *DW*, it is possible to retrieve more than one candidate cases from one single old case.	RO_1: (DW=HAM, DL=WQC) RO_2: (DW=DXB, DL=WQC) **1.1. HAM**-SIN-SYD-**WQC** **1.2.** CDG-**HAM**-SYD-**WQC** **1.3. HAM**-DXB-SYD-**WQC** **1.3.** HAM-**DXB**-SYD-**WQC**
2. Time-related: Current-Weekday (CW), Current-Time (CT) 2.1. If *Index 1.1* matched a Path: *CW* is the same as the Leg_1 Weekday (SW_1) and *CT* is chronologically before the solution's RDT_1. 2.2. If *Index 1.2* matched a Sub-Path: *CW* is the same as the Leg_x Weekday (SW_x) and *CT* is chronologically before the solution's RDT_x (where x stands for the solution leg index matching CW). 2.3. If *Index 1.1* or *Index 1.2* matched: *CW* is one day before the Leg_1 Weekday (SW_1), meaning that cases with a route departing the day after *CW* are also considered.	RO_1: (DW=HAM, DL=WQC) RO_2: (DW=DXB, DL=WQC) CW=Fri, CT=10:00 **2.1. HAM**-SIN-SYD-**WQC** Leg1: **WD1=Fri, RDT1=12:00** Leg2: WD2=Fri, RDT2=22:00 Leg3: WD3=Sat, RDT3=03:00 **2.2.** CDG-**HAM**-SYD-**WQC** Leg1: WD1=Thu, RDT1=23:00 Leg2: **WD2=Fri, RDT2=11:00** Leg3: WD3=Fri, RDT3=22:00 **2.3. DXB**-SIN-SYD-**WQC** Leg1: **WD1=Sat, RDT1=12:00** Leg2: WD2=Sat, RDT2=22:00 Leg3: WD3=Sun, RDT3=03:00
3. Support-indexes: Part-ID (PNR), Priority (PRY), Customer-ID (CID) 3.1. If *Index 1.1* or *Index 1.2* matched: *PNR* is the same as the solution's *PNR*, providing for another case dimension (strengthen similarity metric). 3.2. If *Index 1.1* or *Index 1.2* matched: *PRY* is the same as the solution's *PRY*, providing for another case dimension (strengthen similarity metric). 3.3. If *Index 1.1* or *Index 1.2* matched: *CID* is the same as the solution's *CID*, providing for another case dimension (strengthen similarity metric).	RO_1: (DW=HAM, DL=WQC) RO_2: (DW=DXB, DL=WQC) PNR=9009, PRY=AOG, CID=3001 **3.1. HAM**-SIN-SYD-**WQC** **PNR: 9009** **3.2. HAM**-SIN-SYD-**WQC** **PRY: AOG** **3.2. HAM**-SIN-SYD-**WQC** **CID: 3001**

Table 3.1: Case Indexes Description

3.2.3 Case Matching

Case retrieval techniques search cases in the case library by looking in their defined indexes in order to select a set of cases that partially match the input case. This process is illustrated in Figure 3.3 where a sample set of cases is partially matched.

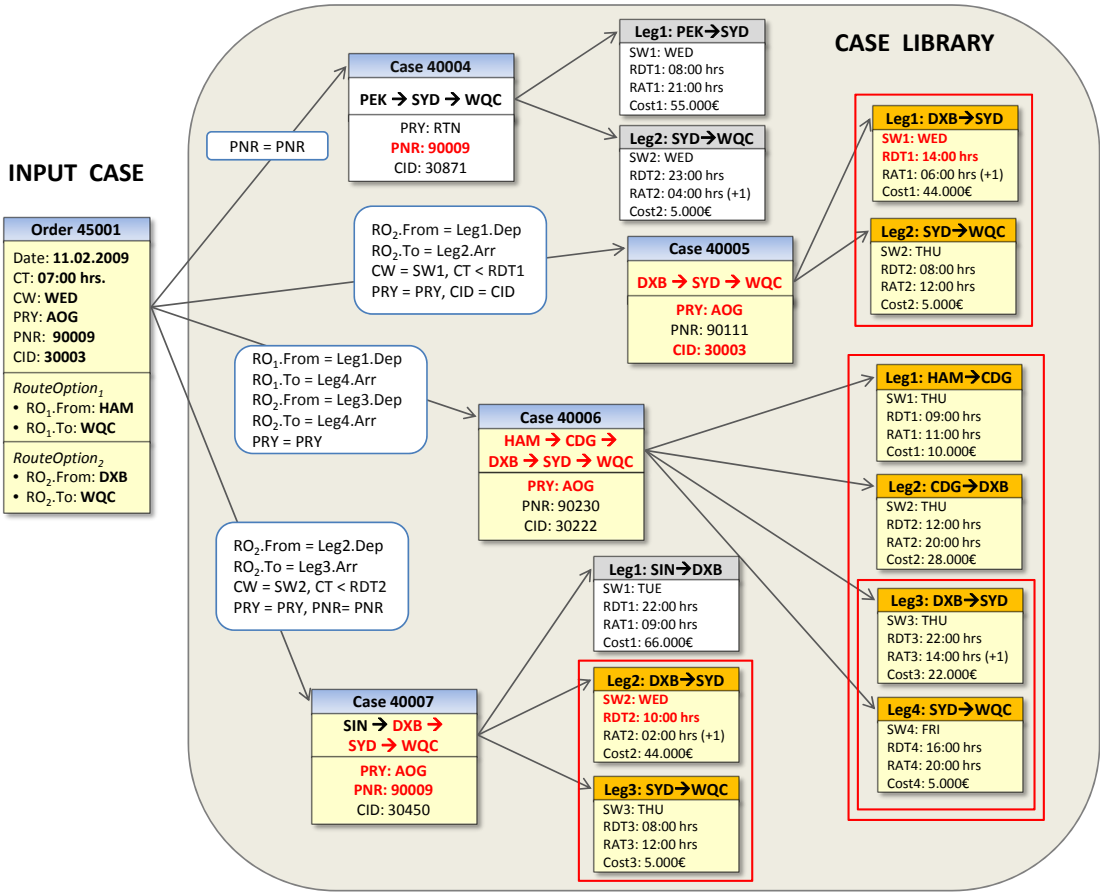

Figure 3.3: Sample Diagram of Matching Process

From this diagram it is possible to identify the attributes that are being compared between cases for performing the matching process (i.e. the white rounded squares describe the matching attributes). Not all attributes have an equal weight and this constrain the cases to be selected. For instance between Order 45001 and Case 40004 there is a match in the *PNR* attribute, but this isolated match does not provide enough information to conclude that the *Solution Path* of Case 40004 can be reused for Order 45001. Therefore, this case is not retrieved because no other attributes match with the input case.

Case attributes usually can be categorized among the following concepts:

- **Features/Descriptors:** these are attribute-value pairs that describe the most relevant aspects of the case description for the matching process. These attributes can be found either in the situation description or the solution description.

  For instance, to confirm that the solution description of Case 40005 is *useful* for the reasoner and therefore retrieved for later ranking, a possible solution route must be firstly matched. The following statements describe how the Order 45001 is matched to Case 40005 for the reuse of its solution path:

$$CPO_{45001}.PRY = CPO_{40005}.PRY$$
$$CPO_{45001}.RO_2.From = CPO_{40005}.Leg_1.Departs$$
$$CPO_{45001}.RO_2.To = CPO_{40005}.Leg_2.Arrives$$
$$(CPO_{45001}.RouteOption_2 \leftarrow CPO_{40005}.SolutionPath_{DXB-WQC})$$

- **Dimensions:** a case dimension is identified *only* when the primary case features have been successfully matched and an attribute-value pair strengthens or weakens the similarity of this matched case against the input case.

  For instance, when matching Order 45001 and Case 40007, the PNR attribute match is relevant for the inference process *only* because the following features have been previously matched:

$$CPO_{45001}.RO_2.From = CPO_{40007}.Leg_2.Departs$$
$$CPO_{45001}.RO_2.To = CPO_{40007}.Leg_3.Arrives$$
$$(CPO_{45001}.RouteOption_2 \leftarrow CPO_{40007}.SolutionSubPath_{DXB-WQC})$$

  The presence of a matching PNR provides the case with another dimension strengthening the inference that the $SolutionSubPath_{DXB-WQC}$ of Case 40007 is a good solution for Order 45001, primarly because both routes are shipping the same part.

  Other attributes that bring dimensions for cases are Current-Weekday (CW), Current-Time (CT), Customer-ID (CID), and Priority (PRY).

- **Multiple Matches:** the case indexes description shown in Table 3.1 considers the possibility that an old case's solution path encloses also a set of possible subpaths to be considered as solution candidates.

  Given also that the situation description of an input case can encompass several *RouteOption* sets (i.e. pair of dispatch warehouse and destination

location) because the part might be available at many different warehouses, then it is possible to find as much solution routes in an old case as many *RouteOption* sets found in the input case.

Let us consider a match between Order 45001 and Case 40006 from Figure 3.3. Order 45001 has two sets of *RouteOption* given that the PNR 90009 is to be found in sufficient amount in warehouses HAM and DXB. When matching against Case 40006 it is found that $RouteOption_1$ matches the original $SolutionPath$ going HAM→WQC and that $RouteOption_2$ matches a $SolutionSubPath$ going DXB→WQC. This means that Case 40006 will provide two candidate solutions for Order 45001.

The case matching is done by the following criteria:

$$CPO_{45001}.RO_1.From = CPO_{40006}.Leg_1.Departs$$
$$CPO_{45001}.RO_1.To = CPO_{40006}.Leg_4.Arrives$$
$$(CPO_{45001}.RouteOption_1 \leftarrow CPO_{40006}.SolutionPath_{HAM-WQC})$$

$$CPO_{45001}.RO_2.From = CPO_{40006}.Leg_3.Departs$$
$$CPO_{45001}.RO_2.To = CPO_{40006}.Leg_4.Arrives$$
$$(CPO_{45001}.RouteOption_2 \leftarrow CPO_{40006}.SolutionSubPath_{DXB-WQC})$$

After the matching, possible feature dimensions are to be searched to have more accurate information of each solution candidate. Once all possible candidates are extracted from all the retrieved and matched cases, the Route Suggester must determine which candidate fits best to the new situation, therefore a similarity ranking technique must be implemented.

3.2.4 Case Ranking

Ranking is the process of ordering partially-matched cases according to usefulness to determine which of them fits best to the new situation. Usually, matching schemes result in the computation of a score that specifies the degree of match between one case and another. The match score can be absolute or relative. An absolute match score is computed independent of other cases, while a relative match score requires comparison to other cases.

The following activities are necessary for implementing a ranking scheme:

- **Finding Correspondences:** this is done to determine which features of a new situation should be matched to which features in a stored situation.

The Route Suggester finds correspondences by determining which features can play different functional roles, and use their values for the roles needed. For instance, locations can either play the role of a dispatch warehouse, an intermediate stop, or a final delivery destination. Therefore, whenever a sub-path of an old case's solution path can serve as a solution for a new case, the roles of these features must be corresponded to fit the new situation. Table 3.2 shows the correspondence between a new case and an old case with a route path and sub-path as matching solutions.

NEW CASE		**OLD CASE**	
Feature	**Value**	**Feature**	**Value**
Order-ID	40007	*Order-ID*	45001
PNR	90009	*PNR*	90009
CID	30003	*CID*	30222
PRY	AOG	*PRY*	AOG
Route Option 1		**Solution Path**	
RO1.From	HAM	*Leg1.Departs*	HAM
RO1.To	WQC	*Leg4.Arrives*	WQC
CW	Wed	*SW1*	Wed
CT	07:00hrs	*RDT1*	11:00hrs
Route Option 2		**Solution SubPath**	
RO2.From	DXB	*Leg3.Departs*	DXB
RO2.To	WQC	*Leg4.Arrives*	WQC
CW	Wed	*SW3*	Thu
CT	07:00hrs	*RDT3*	22:00hrs

Table 3.2: Sample correspondence between two cases

- **Computing Degree of Similarity:** once it is known which features correspond to each other, the degree of similarity between these features can be computed. Again, several methods exist for computing this.

  As mentioned in the first chapters, the goal of both the Route Planner and the Route Suggester is to suggest the route that takes the shortest time from a dispatch warehouse to the spares delivery destination. Based on these criteria, a quantitative scale for similarity fits best the reasoner's goal.

  A quantitative scale refers to a numeric evaluation function that uses relative importance of case's features and the degree of match of each to compute a match score. The function chosen for this evaluation is the *Nearest-Neighbor Matching* (NNM).

In NNM, every feature in the new case is matched to its corresponding feature in the old case; the degree of match of each pair is computed, and added to an aggregate match score. Once all retrieved cases have their match score computed, the most similar case will be the one with the highest match score. The steps to compute NNM are described in Box 3.

```
1. For each feature in the new case:

     a.  Find the corresponding feature in the stored case
     b.  Compare the two values and compute a degree of match
     c.  Multiply by a coefficient representing the importance of the feature

2.  Add the results to derive an aggregate match score
3.  Select the case with the highest score
```

Box 3: Nearest-Neighbor Matching Algorithm

Ranking Guided Example

Recalling the case matching process example for the CBR Route Suggester depicted in Figure 3.3, the Order 45001 is being compared against the case library, which in this reduced sample comprises four past cases. During the case matching process, only three of these cases are partially matched against the input case due to the correspondences found among them. The white rounded squares in the figure show the criteria used to match each particular case.

Therefore, a set of four possible routes are extracted from the three partially matched cases. It must be recalled that Case 40006 matches two possible solutions for the input case. The red frames in Figure 3.3 represent the four *solution candidates*, whose attributes are detailed in Table 3.3.

An empirical ranking approach would easily select the solution candidate that offers the minimum TTT, thus satisfying the goal of the reasoner. However, the ranking evaluation is in direct relationship with the input case information, meaning that even if a solution candidate seems to provide the best route timing, this is irrelevant if not evaluated together with the input case.

Candidates Attributes Summary				
Feature	**SC1**	**SC2**	**SC3**	**SC4**
Order-ID	40005	40006	40006	40007
PNR	90111	90230	90230	90009
CID	30003	30222	30222	30450
PRY	AOG	AOG	AOG	AOG
Departs	DXB	HAM	DXB	DXB
Leg-ID	1	1	3	2
SW	Wed	Thu	Thu	Wed
RDT	14:00	9:00	22:00	10:00
Arrives	WQC	WQC	WQC	WQC
Leg-ID	2	4	4	3
SW	Thu	Fri	Fri	Thu
RAT	12:00	20:00	20:00	12:00
TTT	1320	2100	1320	1560
LegCount	2	4	2	2
Cost	€49.000	€65.000	€27.000	€49.000

Table 3.3: Candidates Attributes Summary

As it can be seen in Table 3.4, when the input case information is related to the solution candidates, derived data is calculated such as the Total Shipping Time (TST) which adds the time to be waited before these routes depart.

Candidates TST			
SC	*TTT*	*TTD*	*TST*
1	1320	420	1740
2	2100	1560	3660
3	1320	2340	3660
4	1560	180	1740

Table 3.4: Candidates Total Shipping Time

From this derived information it can be inferred that $SC1$ and $SC4$ share the same final delivery date/time because they share the same final transportation leg, even when their route schedules are different. $SC2$ and $SC3$ also share the same delivery date/time, but this is because they traverse the same route schedule because $SC3$ is a sub-path of $SC2$. Given this initial ranking based on shipping time, Candidates 1 and 4 are selected among the other candidates.

However, other case attributes that could give a more detailed conclusion are still not being considered. Let us keep these two candidates as the best possible solutions. Then, a differentiation factor must be applied for finding the most accurate solution candidate.

Both cases PRY is AOG just as the input case, so this cannot be the factor. The $SC1$ matches the CID, so both the input and the recorded case are requests from the same customer, meaning that if the customer agreed to traverse through this route once, it is likely to accept it as a solution for a new similar case. Moreover, the $SC4$ does not match the CID, but matches the PNR with the input case. When old and new case share the same PNR it is affirmed that the route operators for the old solution will likely transport the new order because it actually is the same product and no further checks are needed.

Because this is a stronger reason to believe a route will function as it did for a past solution, then $SC4$ is selected as *solution suggestion* for Order 45001. The initial case-based inference process for suggesting a solution ends here, but some case adaptation to actual schedules might be needed.

The similarity ranking inference done in this sample is a straightforward process for a human to solve given the contraints. However, for an automated reasoner, it is necessary to implement an evaluation function that encompasses this ranking process.

Ranking Evaluation Function

The NNM algorithm described in Box 3 suggests that for each feature matched among cases a degree of match must be computed and multiplied by a coefficient that represents the importance of each feature. The results must be aggregated in a match score. The case with the highest score is the most similar to the input case. Therefore, a way to calculate this *aggregate match score* (AMS) must be expressed (3.1).

$$AMS_{case} = \frac{\sum_{i=1}^{n} w_i \times sim(f_i^I, f_i^R)}{\sum_{i=1}^{n} w_i} \tag{3.1}$$

where w_i is the importance of feature i, sim is the similarity function, and f_i^I and f_i^R are the values for feature f_i in the input and retrieved cases respectively.

For computing the degree of match, an importance coefficient for each relevant feature must to established. Table 3.5 shows the scale used to measure the attributes importance.

Importance Scale	
Very High	1.0
High	0.8
Moderate	0.4
Low	0.2
Not assigned	0.0

Table 3.5: Attributes Importance Scale

The indexes defined for case matching will be the set of attributes to consider for ranking. If the PNR attribute matches it will get a value of 1.0, if there is no match then value will be 0. For CID the match will be 0.2, and no match is 0.

If CW matches and CT is before RDT_1 then value will be 1.0; if CW is one day before the solution's first leg weekday SW_1, the value will be 0.4; else, importance will be 0. For PRY if the match is AOG then value will be 0.8; for RTN the value 0.4 will be assigned.

In addition to the indexes, a factor called *Route Quality* (RQ) is calculated for supporting the ranking process. RQ assigns a value between 0 and 1 to each candidate's route and measures how *short* a route schedule is. Formula (3.2) shows how to derive this factor. RQ depends on two other factors: *Transport Time Factor* (TTF) explained in (3.3); and the *Time To Depart Factor* ($TTDF$).

$$RQ_{route} = TTF_{route}/TTDF_{route} \tag{3.2}$$

$$TTF_{route} = TET_{route}/TT_{route} \tag{3.3}$$

To derive the TTF of a route, it is necessary to previously obtain its *Transport Expected Time* (TET_{route}) which is explained in (3.4).

$$TET_{route} = \sum_{leg=1}^{n} TET_{leg} \tag{3.4}$$

TET_{leg} is the value of the expected amount of time that a specific transport schedule should take to perform. This value for each possible route connecting two locations is stored in the system database and is established based on experience

data. The sum of these legs' expected times will be TET for the route they compose.

Finally, the resulting value of TTF reflects how close a route performed in time against its expected established time. Routes that where traversed closer to the expectation get a higher rank value, while routes that are more distant from the expectation receive a lower rank value.

Moreover, $TTDF$ for a route is formulated in (3.5), where TTD is the *Time to Depart* for this route; while MWT stands for *Maximum Wait Time*, a control value established by the Order Desk. For the Route Suggester this value is set to 2880 minutes, meaning that the Order Desk must not take more than 48 hours to determine a shipping route for an AOG situation. The resulting value must then be subtracted to 1. This it is assured that the lower waiting times have a higher rank value, and the longer times have a lower rank value.

$$TTDF_{route} = 1 - (TTD_{route}/MWT) \tag{3.5}$$

Table 3.6 shows the results for the *Route Quality* calculation for each of the routes offered by the solution candidates discussed in the previous section.

Route Quality				
Factor	**SC1**	**SC2**	**SC3**	**SC4**
TET	1250	1870	1250	1250
TT	1320	2100	1320	1560
TTF	0.94	0.89	0.94	0.81
TTD	420	1560	2340	180
MW	2880	2880	2880	2880
TTDF	0.85	0.45	0.18	0.93
RQ	**0.8**	**0.4**	**0.17**	**0.75**

Table 3.6: Route Quality values for each Case

Another factor considered for the Ranking process is the *Leg Count* of candidate routes. The rationale for its inclusion comes from the idea that as the number of transport legs increases the risk probability of having a delay or missing a connection also increases, due to the variability of service levels of transport operators and traversed locations. Table 3.7 depicts the importance values regarding the *Leg Count* factor. More than 4 legs receive 0 as importance value.

1 Schedule Leg	1.0
2 Schedule Legs	0.8
3 Schedule Legs	0.4
4 Schedule Legs	0.2
5 Schedule Legs	0.0

Table 3.7: Leg Count Importance Scale

Once all importance values for attributes and factors are assigned, the defined evaluation function can be computed. Table 3.8 shows the degree of match calculations summary. The *Maximum Importance* column shows the maximum values that each feature can be assigned to. The other columns show the degree of match between features of each solution candidate and the new case's features.

		Candidates Degree of Match				
Feature	**Type**	**Max.Imp.**	**SC1**	**SC2**	**SC3**	**SC4**
Part-ID	Index	1.0	0.0	0.0	0.0	1.0
Weekday/Time	Index	1.0	1.0	0.4	0.4	1.0
Priority	Index	0.8	0.8	0.8	0.8	0.8
Customer-ID	Index	0.2	0.2	0.0	0.0	0.0
Route Quality	Factor	1.0	0.8	0.4	0.17	0.75
Leg Count	Factor	1.0	0.8	0.2	0.8	0.8
Aggregate Match Score		5.00	3.60	1.80	2.17	4.35
Normalized Match Score			.720	.360	.434	.870

Table 3.8: Candidates Degree of Match Summary

The row before the last shows the *aggregate match score* resulting from the evaluation function. The bottom row normalizes each candidate's match score in respect to the maximum possible match score. The highest AMS is .870 corresponding to *Candidate 4* (SC4) which is the case to be selected as solution for the previously discussed sample problem.

The use of this numeric evaluation function enables the automation of the ranking process. However, when similarity scores among candidates is close, it can mean that each solution is about equally similar to the new situation. There are two possible explanations for such situation: either the cases are indeed quite similar to one another, or the matching criteria have not been chosen well. If this situation is recurrent, then it is suggested to collect more cases within the case library to allow a more balanced set of suggestions or some better means of distinguishing cases must be found (i.e. better similarity or matching criteria).

3.2.5 Case Revision

The revision of a suggested solution obtained from an old case is necessary to prove that it actually fits the new situation. Because no old situation is ever exactly the same as a new one, adaptation is a necessary process in CBR.

Adaptation might be as simple as substituting one component of a solution for another or as complex as modifying the overall structure of a solution. There are many adaptation techniques performing diverse updates in cases. The Route Suggester uses a type of substitution method called *reinstantiation*.

Substitution is the process of choosing and installing a replacement for some part of an old solution. Moreover, reinstantiation is used when the frameworks of an old and new problem are obviously the same, but roles in the case are filled differently than roles in the old one. This means that the data bindings of the old solution components are replaced by new data bindings.

Specifically, the adaptation done by the Route Suggester is to bind the suggested solution route to actual scheduled transport links stored in the system's database. This schedule information is constantly fed by transport operators. The reliability on availability of updated transport links that match the suggested route is based on the belief that operators maintain a constant schedule for the routes they serve.

Whenever a transport link schedule is unavailable due to seasonal or operational changes, the adaptation process on the suggested case will not be successful. In this situation, the next solution candidate with higher ranking will then be selected and again sent to adaptation. If none of the possible candidates can be linked to an actual schedule, the Route Suggester presents no solution and the candidate cases are dismissed from the case-library. This way the case-library is maintained up to date in regard to actual operator's schedules.

Finally, the new case is stored and indexed in the case library for future use. The case comprises the situation description and the detailed solution, plus any support data for presentation or calculation purposes.

3.3 Integration of Routing Approaches

After describing the Route Suggester it is possible to understand that the case-based reasoning approach optimizes the route finding problem and enhances the possibility of learning.

However, the CBR approach cannot completely replace the Route Planner based on Dijkstra's algorithm, because it lacks of a route searching functionality. Also, the Route Planner itself is not the optimal solution in terms of computation time. This means that the single implementation of any of these approaches is probably not the best solution.

Dijkstra's algorithm is the only technique that is able to work independently to solve the problem. The algorithm searches in a weighted directed network and it finds the shortest path from a source to every other node in the network.

Although this algorithm is efficient, searching the whole route network with thousands of nodes and edges to find the shortest route will take a long time. The problem with Dijkstra's algorithm is that it is not always necessary to search through the whole route network in order to find the solution. For example, if an order must find a route going PEK→SYD (Peking to Sydney) most probably one will not consider those routes leading to destinations in Europe or America.

Case-based reasoning has the advantage of learning as it acquires more cases. For the route finding problem, many routes could be previously stored, thus providing a robust case library for problem solving. However, it is highly inefficient to store all the possible routes from one location to another for a huge network. This means that if all possible routes cannot be stored, then there will not always be old cases to fit new situations. In such cases the only possible solution is to use a route searching algorithm.

To summarize, the following are the advantages and disadvantages of the described solution approaches:

- **Route Planner:** Dijkstra's algorithm is able to find the best solution but this is done by a blind search which can be time consuming and wasteful in terms of computation.
- **CBR Route Suggester:** CBR gives a solution to a new problem instantly if there are similar cases in the case library, but if there are no case matches for this situation it will not provide a solution.

Therefore, it can be concluded that integrating both approaches is a better solution than any of them as a single. In an integrated approach, for example, when there is an old case in the case library which matches a new incoming case, there will be no need to search with Dijkstra's algorithm, but simply output the route provided by the CBR Route Suggester and adapt it to the current schedule information.

Moreover, when a similar case is not found, the CBR Route Suggester can pass the problem to the Route Planner for finding a route through a network search. Once a route is found, this can be added to the case-library improving the chances of solving a future problem with an old solution, gradually minimizing the use of the Route Planner.

This integrated approach is described as an algorithm in Box 4 suggesting a sequence of problem solving: the CBR approach should try to solve the problem first because it is more efficient, but if it fails, the problem will then be passed to the Route Planner. Finally, after using any of the approaches, the case must be properly indexed and recorded in the case library.

```
1.  Input a new situation (incoming order)

2.  Execute CBR Route Suggester to find similar cases.

3.  If any similar cases are found:
       Then:  Select case solution with best match score.
              Adapt solution with new schedule information.
       Else:  Execute Route Planner.

4.  Add new case to library with proper indexes.
```

Box 4: Integrated Solution Approach Algorithm

Chapter 4

Web-Based Integrated Solution

4.1 Introduction

The last part of the project is to make the functionality of the Route Planner and the CBR Route Suggester available through a common interface. This interface intends to provide the Spares Order Desk with a decision-support tool that can be accessed to assess incoming AOG situations in a single application.

Therefore, a Web-based application was designed and developed to provide the system with online availability via the World Wide Web. The foundations to create such an application are:

- The integrated system can be easily accessed via a Web browser within organizational Intranet boundaries, and its contents are dynamically generated by user request.
- The application is developed, hosted and executed using reliable open-source technologies and there is no need of installing any software within the user's environment.
- The system can be easily integrated to any platform due to the high level of interoperability provided by the implementation done in a Java-based environment.
- The system architecture is based on a multi-tiered architecture allowing modularity and customization through distributed computing.

4.2 System Architecture

As stated above, the Web-based application was developed using a multi-tiered architecture. Apart from the usual advantages of modular software with well defined interfaces, the multi-tier architecture is intended to allow any of the tiers to be upgraded or replaced independently as requirements or technology change. The current system architecture is depicted in Figure 4.1.

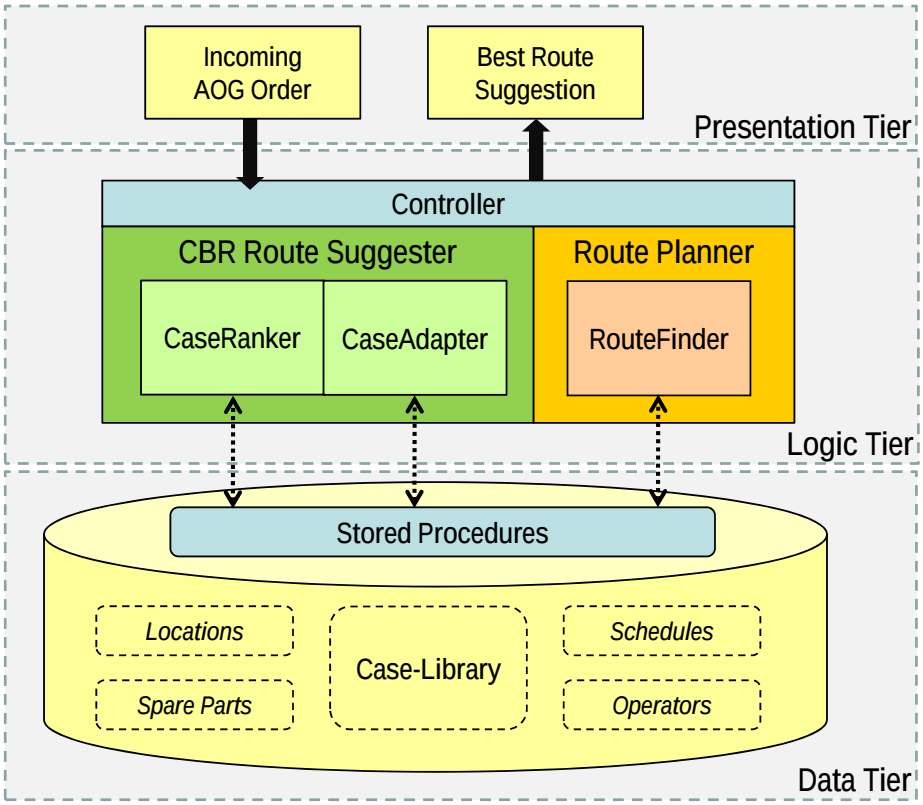

Figure 4.1: System Architecture

4.2.1 Data Tier

This tier consists of a Relational Database Management System (RDBMS) where the data is kept neutral and independent from the application. The data separation from other tiers provides easier scalability and performance improvements.

The system database model defined in Figure 2.3 is the repository represented by this tier. This database, maintained in a MySQL 5.0 management system, contains the following: the collection of previous cases organized in a Case-Library; the catalogue of Spare Parts, Locations, Operators and Schedules; and a set of Stored Procedures which contain all the queries and methods for data access.

4.2.2 Logic Tier

The Logic Tier controls the application's functionality by performing the data processing. This tier contains the implementations of the Route Planner and the CBR Route Suggester based on the Java Platform. Also in this tier, a CBR support tool was used to graphically represent the data-flow during the best-route decisioning process, this is the Erudine Behavior Engine [21].

CBR Support Tool: Erudine

Erudine presents a user interface in which it is possible to define a *Knowledge Model*. This model is comprised by a set of interconnected nodes representing the different data system states during the processing of an Order. The data sets to be used as inputs and outputs can automatically be pulled or pushed respectively from the defined relational database. A sample Erudine's Knowledge Model for the project is represented in Figure 4.2.

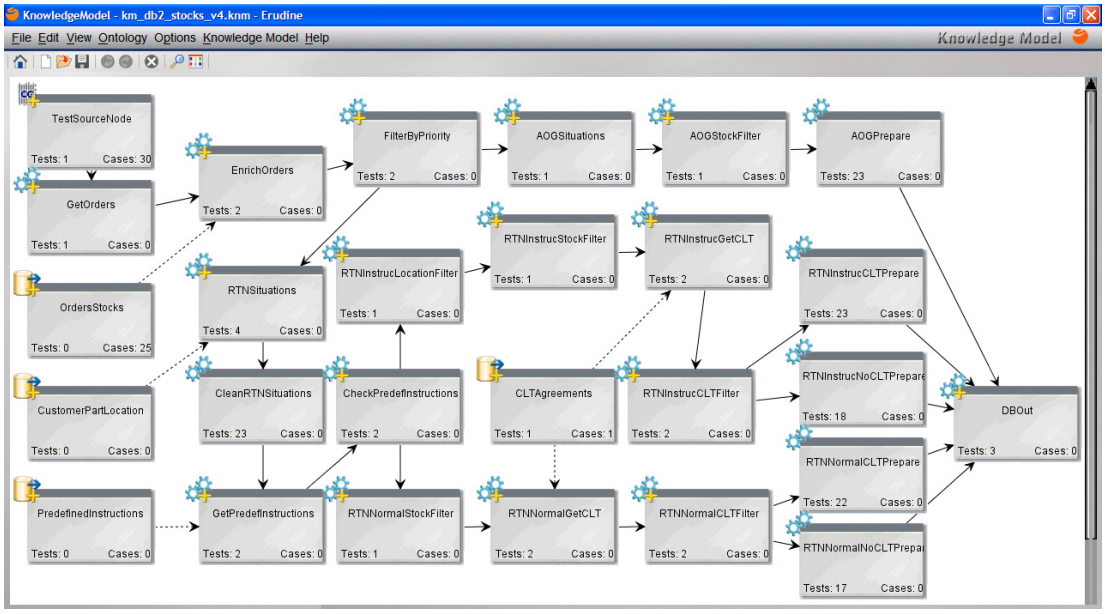

Figure 4.2: Erudine's Knowledge Model Sample

Within these nodes, it is possible to establish rules to evaluate data states in order to direct/redirect the flow of information among the model nodes. A detailed view of a node with a basic rule is shown in Figure 4.3. In this node it is defined that only Warehouses with enough product stock availability are to be considered as possible Dispatch Warehouses for the route determination problem.

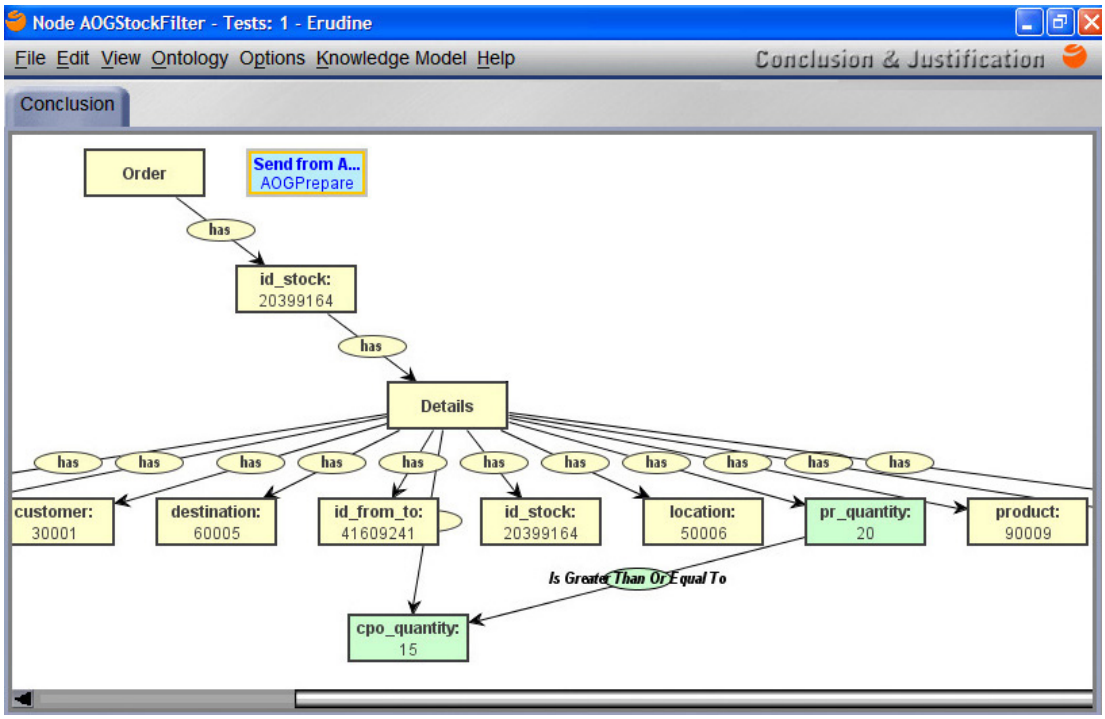

Figure 4.3: Erudine's Knowledge Node Sample

For testing the validity of the model, corresponding sets of input cases and output solutions are pushed into the model. The model processes them and relates their relevant features based on those mentioned rules, in order to help the system designer to identify possible inferences to be made for modeling a CBR system. This way, the logic can be continuously refined until many of the possible cases are covered. This is a very powerful way to efficiently understand and represent the domain-knowledge. Erudine was used to support the logical design of the case matching process for the Route Suggester.

Java-based Processes Implementation

After the system design was completed, practical implementations of the Route Planner and the Route Suggester were developed using the Java platform. The decision to choose the Java platform is because it is a standardized and mature technology with vast learning resources and huge interoperability possibilities with other platforms.

For the Web application development the specific technology used is the Java Server Pages (JSP). JSP enables software developers to create dynamically-generated web sites in response to a client request, allowing Java code to be embedded into static Web content [10] [23].

Initially, a *request* is received via a Web browser (i.e. user selects an Order for decision-support). The responsible JSP for catching this action is sent to a compiler which generates an object that dynamically processes requests and construct responses, also called Java Servlet. The Servlet executes the proper methods indicated by the JSP, which might need to query/save information from/to a relational database. Finally, a *response* is generated containing the execution results formatted in HTML and sent back to the browser for presentation. Figure 4.4 depicts a typical JSP architecture.

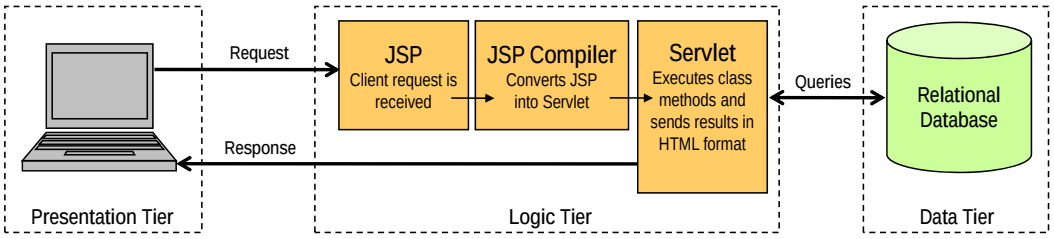

Figure 4.4: Typical multi-tiered JSP Architecture

For the Java-based application design, a *class diagram* is defined to visualize the system classes and the methods they implement. Figure 4.5 shows an integrated class diagram for the Route Planner and the CBR Route Suggester. The Integrated Approach main classes are the following:

- *Controller* is an implementation of a process dispatcher that either calls the *CaseRanker* to get a case solution suggestion or the *RouteFinder* for route finding.
- *RouteFinder* is an implementation containing the Route Planner's search methods.
- *Route* is a generalization of the route concept. This is a list of objects representing the nodes that compose a route.
- *Dijkstra* is an implementation of the Dijkstra's algorithm that searches for the shortest path in a weighted oriented graph.
- *IGraph* is an interface that specifies a weighted oriented graph and it is used by the search algorithm in order to deal with graphs.
- *NetworkGraph* is an implementation of the weighted graph.
- *CaseRanker* is an implementation of the matching and ranking process for an inputted order.

- *CaseAdapter* is an implementation of the adaptation process that updates the order's scheduled legs data with current schedules.
- *Order* is a generalization of the order concept. It contains a *SparePart*, a *Customer*, a solution *Path*, and a set of dispatch warehouses and a delivery point that are defined by the class *Location*.
- *SparePart* is a generalization of the spare part concept.
- *Customer* is a generalization of the customer concept.
- *Location* is a generalization of the location concept. It can be instantiated as a dispatch warehouse, a delivery location, a leg departure point, or a leg arrival point.
- *Path* is a generalization of the solution path concept. It contains a set of scheduled legs defined by the class *SchedLeg*. This class has empty values for a new incoming *Order*.
- *SchedLeg* is a generalization of the scheduled leg concept.
- *Operator* is a generalization of the transport operator concept.

Web Application Demonstrator

A demonstrator was developed for testing the Web application's integrated functionality. The following is a list of the components integrating the demonstrator:

- **JAR file:** the Web application is contained in a Java Archive (JAR) file which is the standard for distributing Java classes and associated metadata.
- **JSP files:** they are the dynamic-generated Web pages for realizing requests to the application and presenting results in a Web browser.
- **Configuration file:** a Web configuration and deployment file that defines the application's components, interfaces, and parameters.
- **Knowledge Model file:** the Erudine's knowledge model file defining the decisional rules for evaluating situations. The Erudine Engine JAR file is also attached for the validation and execution of this model.
- **Database Script file:** contains the database structure and system data. It must be initialized and maintained in a MySQL 5.0+ database server.

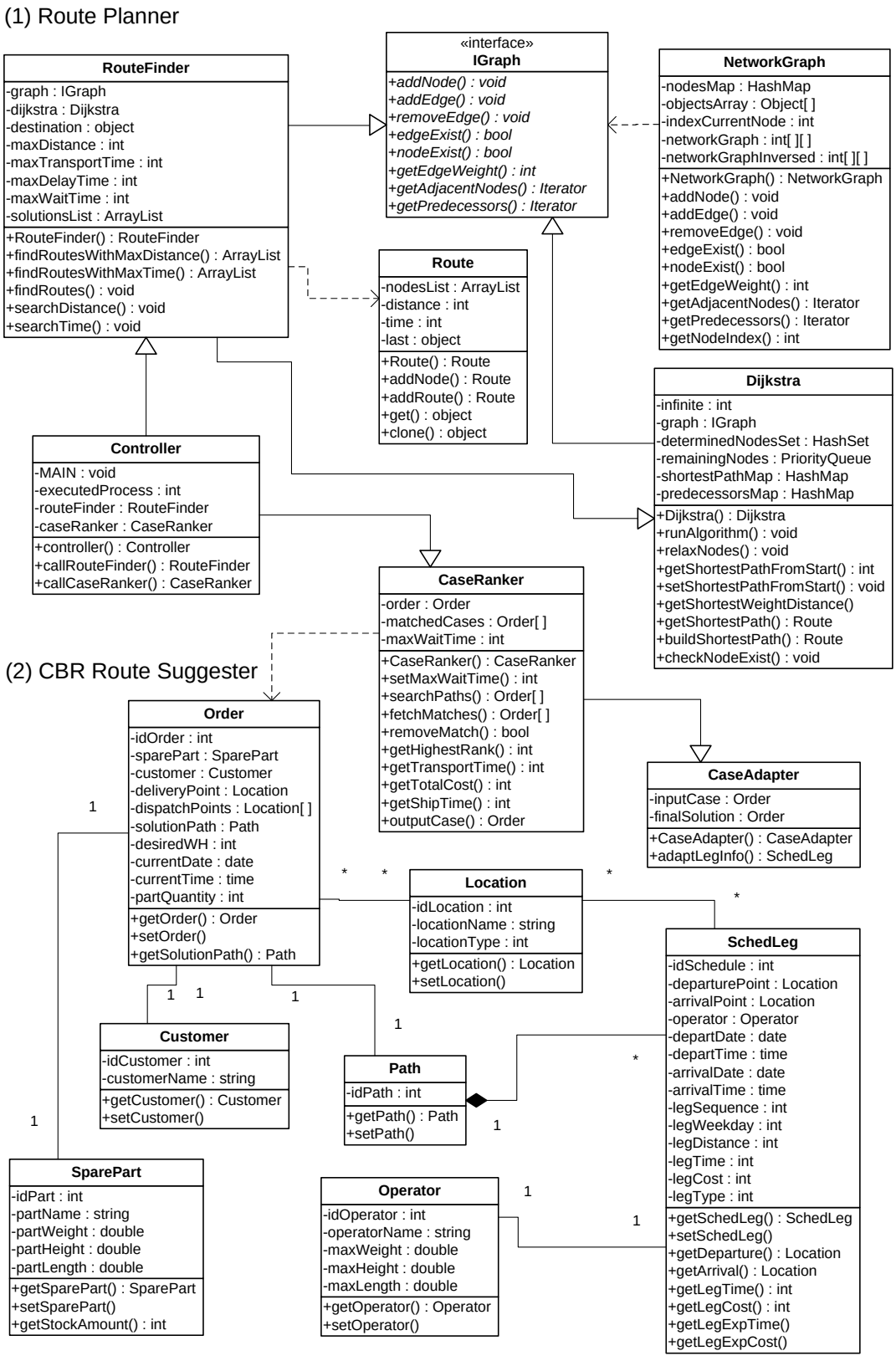

Figure 4.5: Integrated Approach Class Diagram

4.2.3 Presentation Tier

This tier provides a set of interfaces for the Spares Order Desk user, where it is possible to: receive incoming orders, request for CBR solution suggestions or compute the best shipping route, and finally record the confirmed decision. This functionality enables the real-time processing of orders. These interfaces are a sequence of Web pages done in JSP language embedded with HTML formatting.

The Web Interfaces sequence is very simple and is ordered as follows:

1. **Order Selection:** this is the starting point of the sequence (view Figure 4.6). The top section shows the list of orders waiting to be processed, meaning that yet no solution has been selected for them; the bottom section shows the details of the order clicked on the top section, providing a button to query the *CaseRanker* for routing suggestions.

2. **Similarity Check:** this screen shows the results of the *CaseRanker* for the order selected in the first interface (view Figure 4.7). The top section shows the order details (situation description); the bottom section shows the best matching case (solution description), with its similarity ranking value. A button is provided to query the *CaseAdapter* for updated schedule information on this suggestion. If no case matches the order, the *RouteFinder* is automatically queried for performing route search and the view is redirected to the third interface.

3. **Route Suggestion:** this screen shows a best route solution, either outputted by the *CaseAdapter* or by the *RouteFinder* (view Figure 4.8). In the case that the second interface provided a best case match, this screen will show the adapted information of the legs composing the solution suggestion. Else, this screen will show the results of the route search computation done by the *RouteFinder*. A button is provided for the user to confirm that the outputted route is to be selected as the final solution for the order.

4. **Order Processing:** the last interface shows the confirmation that the order was successfully processed (view Figure 4.9). To reach this screen, the user must have confirmed in the previous interface that the outputted route is an acceptable solution. In this step, the order status is set to *processed* and the case is finally recorded in the case-library. A button to return to the first interface is also provided.

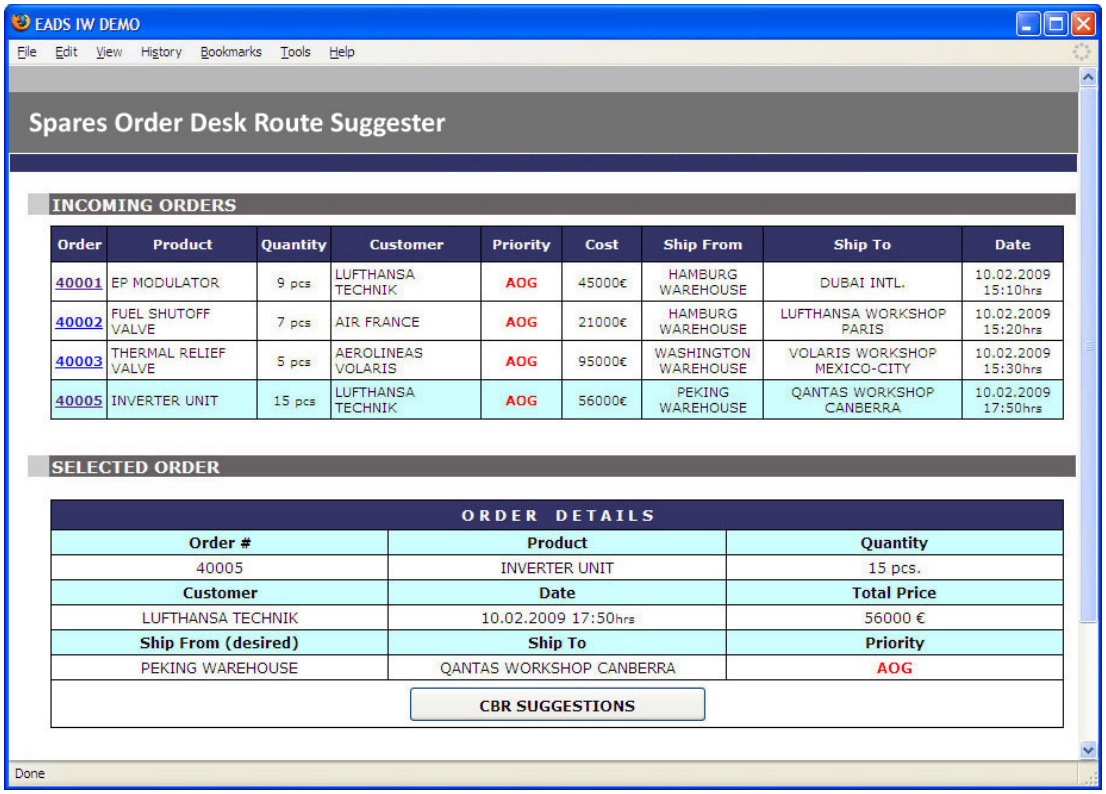

Figure 4.6: Web Interface 1: Order Selection

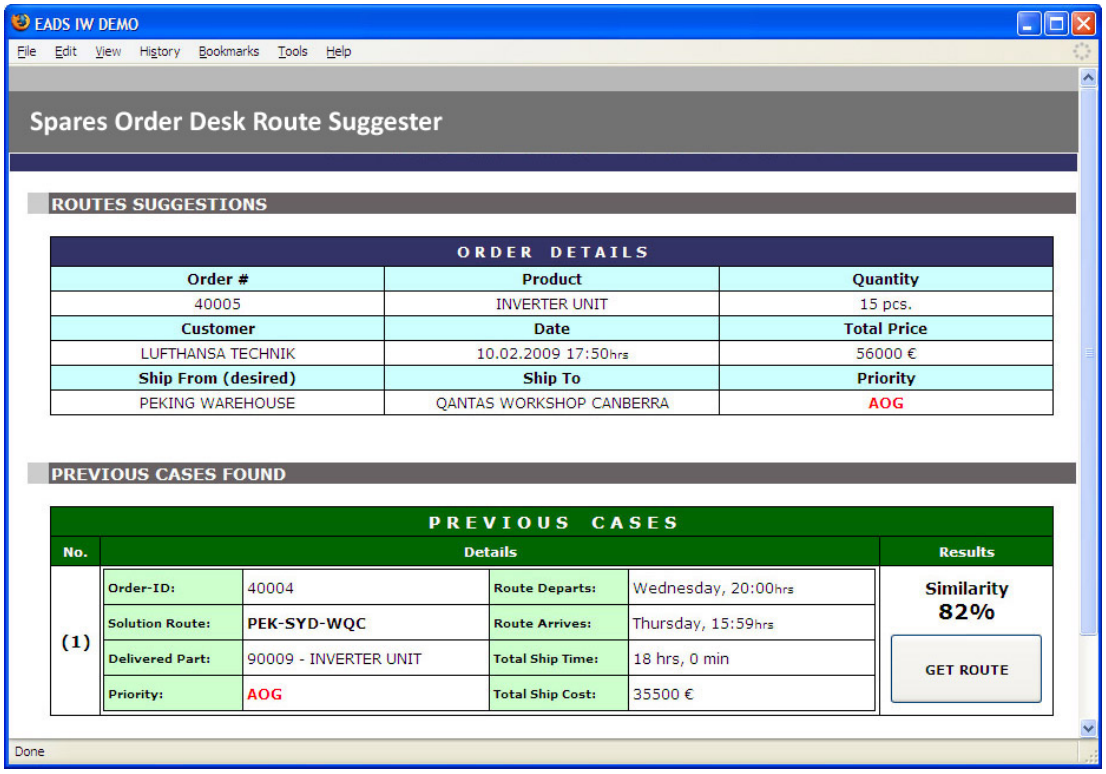

Figure 4.7: Web Interface 2: Similarity Check

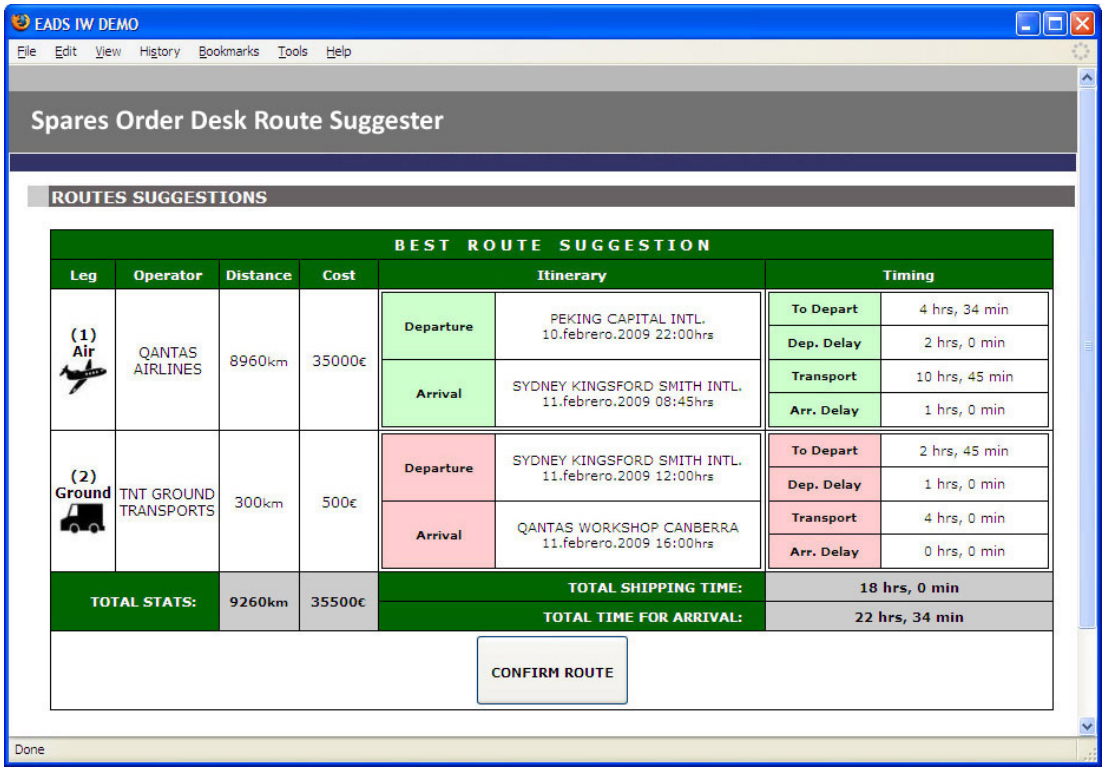

Figure 4.8: Web Interface 3: Route Suggestion

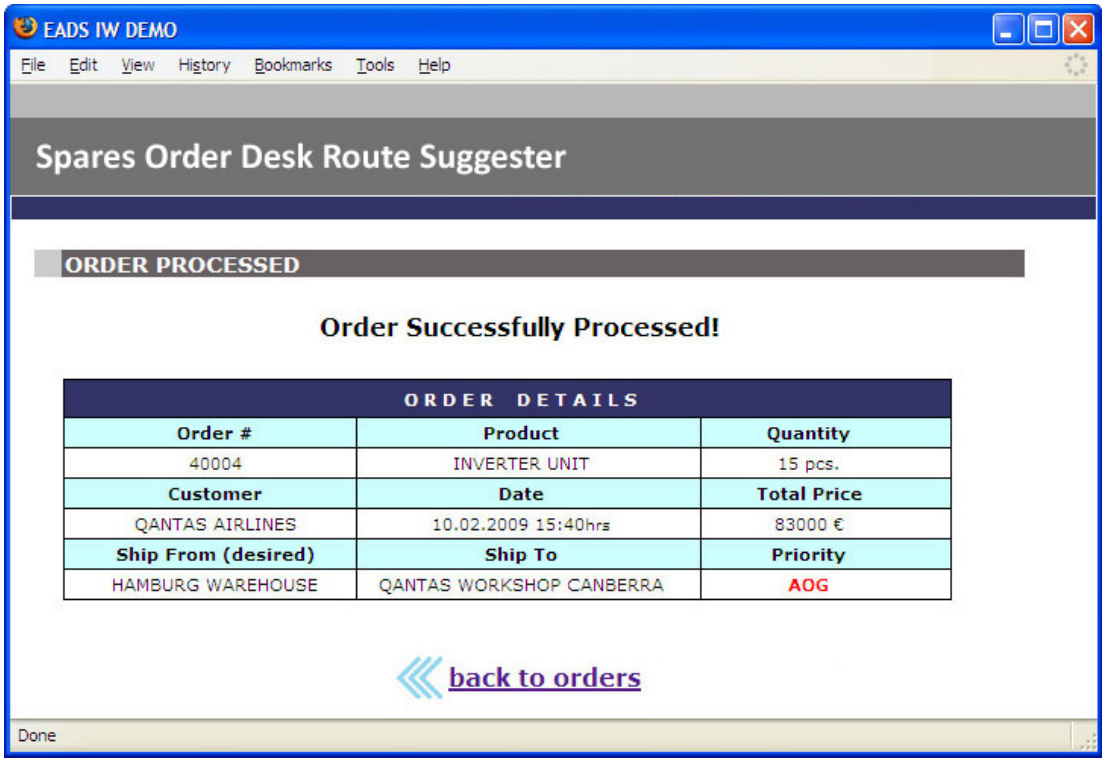

Figure 4.9: Web Interface 4: Order Processing

Chapter 5

Results

5.1 Introduction

In previous sections it was discussed that the solution approach based on Dijkstra's algorithm provides a reliable but not very efficient solution. It was also mentioned that the case-based reasoning approach provides a more efficient solution as long as the case-library is well populated and the retrieval mechanisms were correctly chosen. Moreover, proper measurements are needed to illustrate this in a certain way.

Case-based reasoning is a relatively young field of study which traces its roots from the early 1980s. However, despite increased interest on the field during all these years, yet there are not clear established standards on how a CBR application should be revised for success. This is mainly because every reasoner is different to the others, and the goals they are designed to achieve are so diverse that is almost impossible to establish a standardized method. Nevertheless, the CBR application should efficiently achieve the goal it was made for, namely the determination of a spares supply route.

Since it is difficult to establish a metric for comparing one route to another due to the different contexts in which they are suggested, the only relevant metric we can rely on is the computation time. If the integrated approach determines a spares supply route in considerable less time than the average time the Order Desk takes to find a solution, then it can be concluded that the project could succeed in saving man-hours dedicated to this task.

5.2 Experiments

The goal of the experiments is to show that the proposed solution approaches of this project produce reasonable shipping routes to incoming spares orders in a competitive time. The experiments have been performed on a portion of a real spares transportation network.

The initial data set are 50 locations (nodes) with 2,227 different transportation schedules (edges) connecting them. Their time of departure has been randomly determined to provide the set with more diverse transport legs connectivity. Their arrival time has been determined based on the *expected transport time* values for each specific pair of locations. Delay times information for every location has been randomly generated. These schedules are repeated for a duration of one week, just as flight schedules are regularly planned, meaning that the total set of transportation schedules accounts for 15,589. These schedules have a randomized operator, meaning that not all legs are available for all spare parts. An initial set of 20 spare parts with different measurements enables these variations. There are a set of 5 established supplies warehouses, meaning that each order has the possibility of being dispatched from a maximum of 5 locations. All these randomized information provides for a realistic picture of a typical transport routes network.

Sets of incoming spares orders are subsequently inputted, providing first a case that does not exist in the case-library, and then re-inputting this case directed to the same delivery destination but with different attribute values. This way the Web-Application has means of determining a route either with the Route Planner or the CBR Route Suggester. The Planner was tested with a naive implementation of Dijkstra's algorithm and also with an optimized version of this algorithm using a priority queue with a Fibonacci heap.

Table 5.1 shows the average computation time needed to determine a route for one case tested on an Intel Core 2 Duo T5450 processor with 2GB of RAM.

Tested Approach	Min. Time per Order	Max. Time per Order
Route Planner (naive)	975 sec.	4875 sec.
Route Planner (priority queue)	897 sec.	4485 sec.
CBR Route Suggester	57 sec.	285 sec.

Table 5.1: Average Computation Time for three Approaches

5.3 Conclusions

Looking at the results from these computations it is clear that searching a route is extremely expensive when compared to the case-based reasoning approach. Moreover, when the routes network includes thousands of locations and possible schedules between them, the computation time for the first two approaches is not acceptable anymore. To address this limitation, the route search algorithm could be directed with support data. For instance, locations could be grouped into self-defined geographical zones where the search could focus only on relevant locations enabling a search within a reduced routes network.

The notably lower computation time shown by the CBR Route Suggester is due to the fact that it has lesser search to do than the other algorithms because most of the search was done for the previous cases used. As long as the case-library collects more and more cases, the probability of finding matches will be higher and the performance of the integrated application could increase. Unfortunately, in order to make a projection on how fast could the reasoner learn to improve the application performance, the problem of determining how many routes exist between a start and destination node must be firstly addressed. This problem might be mathematically intractable given the variation and non-completeness of the routes graphs.

The average amount of AOG situations assisted by a Spares Order Desk accounts to nearly 5,000 per year with an average response time of one hour and a half (5400 seconds) [5]. If an AOG situation is to be solved by the CBR Route Suggester given that the case library provides for old matching cases, the Spares Order Desk could save up to 95% of the time required for route determination, yielding in important costs savings for the group. If the problem is to be solved by the Route Planner, still up to 17% of time could be saved if the search problem could be always reduced to a search within a maximum of 50 nodes, just as the experiment found.

However, further testing with real situations within a Spares Order Desk could offer a much clearer estimation of the possible time and cost savings of the project's implementation. Also, feedback from real users regarding the reliability of the suggested routes could provide a more accurate evaluation of the goals achieved by the project.

Chapter 6

Conclusions

6.1 Final Discussion

The project as a whole has the main purpose of proposing a good solution approach for the route-finding problem of the spare parts supply chain within the aviation industry. The activities carried out to complete it and their respective obtained results allow us to conclude that the current integrated solution works properly for suggesting acceptable and realistic routes, thus successfully supporting the decision-making process of the Spares Order Desk.

Currently, the integrated solution approach has been found to be efficient, but it is difficult to give a precise evaluation of the system performance since it is in a prototype phase. A continuous evaluation process could provide us with a better metric for confirming that the reasoner is actually able to replace some of the human reasoning during the decisioning process. During an initial phase, the solution approach proposed in this paper could be parallelly used to the current route determination process in order to measure the reliability of its findings. Nevertheless, the dramatic reduction in terms of computation time confirms one of the many benefits of applying reasoning techniques for decision-support.

Although the approach has mainly focused on logistics for spare parts management, a similar implementation framework and methods could be applied to other engineering disciplines. Reasoning techniques based on expert knowledge are an interesting approach for improving the accuracy of systems interacting with changing environments such as the real world. More exploration of these techniques could assist many industrial processes that rely entirely on human experts.

6.2 Contributions

The project has shown that it enables a faster decisioning for the spares delivery process compared to the current Order Desk route planning process. The main contributions for achieving this goal are the activities defined in the project scope, namely:

1. **Route Planner:** an implementation of the Dijkstra's algorithm for performing route search in a weighted oriented graph. This graph is a representation of the spares delivery transportation network that considers current schedules, time and delay information, and transport operators. The Planner finds the shortest path in terms of time from one location (dispatch warehouse) to another location (delivery destination), outputting a scheduled route for decision support within the spares route determination process.

2. **Case-Library:** a relational data repository that contains the information of processed spare parts orders and the routing solutions selected for them. This library represents the collected knowledge to be used for future situations and for historical evidence of the Order Desk decisions regarding the spares delivery process.

3. **CBR Route Suggester:** an implementation of a case-based reasoner that proposes routes for mitigating AOG situations based on the captured knowledge collected in the case-library. It makes use of particular features of new orders for mapping them to orders already processed, then it selects the best route from the retrieved set of possible solutions based on a ranking mechanism, and finally adapts this route with updated schedule information. These findings based on analogy provide a much faster solution to the spares route determination process.

4. **Integrated Web-Application:** an interface that provides online access to the aforementioned implementations in an integrated fashion. Within this application, incoming spares orders can be evaluated by firstly looking for a reliable delivery route through the CBR Route Suggester, and secondly searching for a route through the Route Planner if the reasoner provided no suggestions. Finally, the results of the findings are outputted to a Web-based interface available via a standard Web browser.

6.3 Shortcomings

During the research done for the project, many possible features and improvements were identified, but due to the reduced timeframe were not possible to implement.

The following are the main identified project shortcomings:

- The CBR approach could extend its ranking criteria to improve the reliability of the suggested routes. For instance, including some of the following knowledge-based features: current weather conditions in locations, ranking of the operators service levels, road conditions and average traffic for ground transportations, customer's feedback on traversed routes, etc.

- Instead of having mutually exclusive approaches, the Route Suggester could work together with the Route Planner to build up routes. For instance, if during the case matching process is found that some legs of a route are missing for completing a reliable solution, the Suggester could ask the Planner to search routes for these particular legs, attach them to the incomplete route, and finally adapt the results to obtain a fully functional route. This combined effort could be implemented both ways.

- The case adaptation process could propose alternative delivery points when routes to the original destination are not found. For instance, it could suggest arriving at a location *close* (in terms of transportation time) to the final location. Including geographical information of locations and knowledge on availability of unscheduled transportation means such as taxis or helicopters, could improve solutions that initially might seem useless and go further into more realistic routes suggestions.

- The quality of the cases stored in the case-library could be enhanced by executing planned maintenance tasks for identifying cases rarely used or cases that are already outdated. This way the case matching mechanism could be faster executed.

6.4 Future Work

The project could be initially extended by implementing the shortcomings mentioned in the previous section. Moreover, aside from the routing problem, there are other identified opportunity areas for the use of case-based reasoning within a Spares Order Desk.

The following are the identified scenarios that could benefit from this approach:

- **Spare Parts Leasing:** commonly, a Spares Order Desk leases tools to customers as another maintenance service. When a tool is leased to a customer, it becomes *invisible* for others when a leasing agreement is signed, enabling a net inventory availability strategy. However, the leased tool sometimes waits some days/weeks in the warehouse until the customer makes use of it. In the case that during this *invisibility* time the tool is urgently needed by a different customer, a reasoning mechanism could propose a logistic plan to fulfill both customers' requests.
- **Spare Parts Alternatives Proposal:** sometimes, when customers order spare parts that are not available in stock, the Order Desk suggests alternatives such as the next higher assembly of the spare part or interchangeable parts that share same fit, form, and/or function. A reasoning mechanism could propose these alternatives based on a defined spare parts ontology.

Furthermore, the principles of routing with a case-based reasoning approach could be used in many different types of applications. There could be interesting applications that fit the everyday human life, such as: route planning within a city providing drivers with suggestions regarding alternative routes to avoid traffic jams or accidents; planning of touristic paths based on the frequency of visitors on sightseeing places within a city; suggestion of self-guided tours in museums based on the recorded experiences of previous visitors; proposal of supermarket efficient-shopping routes based on the location of the groceries and recorded customer habits.

References

[1] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications Vol. 7, no. 1*, pages 39–59, 1994.

[2] Jonathan Allen. Integration of case based retrieval with a relational database system in aircraft technical support. *Proceedings of the First International Conference on Case-Based Reasoning Research and Development*, pages 1–10, 1995.

[3] Air Transport Association. E-business specification for materiels management: Spec 2000 overview. http://www.spec2000.com/10.html.

[4] Ralph Bergmann and Klaus-Dieter Althoff. Methodology for building case-based reasoning applications. *Case-Based Reasoning Technology, From Foundations to Applications*, pages 299–326, 1998.

[5] Commercial Aviation Services Boeing. Commercial airplanes operations center website: http://www.boeing.com/commercial/global/opscenter.html.

[6] Boris Cherkassky, Andrew Goldberg, and Tomasz Radzik. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical Programming Vol. 73*, pages 129–174, 1996.

[7] Edsger Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik Vol. 1*, pages 269–271, 1959.

[8] Michael Fredman and Robert Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the Association for Computing Machinery Vol. 34*, pages 596–615, 1987.

[9] Giorgio Gallo and Stefano Pallottino. Shortest path algorithms. *Annals of Operations Research Vol. 13*, pages 1–79, 1988.

[10] David Geary. *Advanced JavaServer Pages*. Prentice Hall, 2001.

[11] Karen Haigh and Manuela Veloso. Route planning by analogy. *Case-Based Reasoning Research and Development, Proceedings of ICCBR-95*, pages 169–180, 1995.

[12] Michael Hugos. *Essentials of Supply Chain Management*. Wiley, 2006.

[13] Hosang Jung, Frank Chen, and Bongju Jeong. *Trends in Supply Chain Design and Management*. Springer, 2007.

[14] Janet Kolodner. *Case-Based Reasoning*. Kaufmann, 1993.

[15] Bing Liu. Integrating case-based reasoning, knowledge-based approach and dijkstra algorithm for route finding. *Proceedings of the Tenth Conference on Artificial Intelligence for Applications*, pages 149–155, 1994.

[16] John Muckstadt. *Analysis and Algorithms for Service Parts Supply Chains*. Springer Science and Business Media Inc., 2005.

[17] Christopher Riesbeck. *Inside case-based reasoning*. Erlbaum, 1989.

[18] Stuart Russell. *Artificial Intelligence: A Modern Approach*. Prentice Hall Inc., 2003.

[19] Frank Schulz and Dorothea Wagner. Dijkstra's algorithm on-line an empirical case study from public railroad transport. *International workshop on algorithm engineering No. 3, Vol. 1668*, pages 110–123, 1999.

[20] Song-Quan Shi. Study on real time traffic flow routing algorithm based on case-based reasoning. *Intelligent Transportation Systems Proceedings, Vol. 1*, pages 163– 167, 2003.

[21] Ruby Singh. *Introduction to the Erudine Behaviour Engine: Training Guide*. Erudine http://www.erudine.com/introduction-to-the-erudine-behaviour-engine/, June 2008.

[22] Jon Sneyers, Tom Schrijvers, and Bart Demoen. Dijkstra's algorithm with fibonacci heaps: An executable description. pages 182–191, 2006.

[23] Andrea Steelman and Joel Murach. *Java Servlets and JSP*. Mike Murach & Associates, 2004.

[24] J. Toussaint and K. Cheng. Web-based case-based reasoning as a tool with the application to tooling selection. *The International Journal of Advanced Manufacturing Technology Vol. 29*, pages 24–34, 2006.

[25] Ian Watson and Farhi Marir. Case-based reasoning: A review. *The Knowledge Engineering Review Vol. 9*, 1994.

[26] Benjamin Zhan. Three fastest shortest path algorithms on real road networks: Data structures and procedures. *Journal of Geographic Information and Decision Analysis*, 1:69–82, 1997.

[27] Benjamin Zhan and Charles Noon. Shortest path algorithms: An evaluation using real road networks. *Transportation Science*, 32:65–73, 1998.