# VIRTUAL LOADER

## A DEVELOPMENT ENVIRONMENT FOR LOAD PLANNING SYSTEMS

**Jacob Feldman**
Technical Director
Solution Dynamics Inc.
981 US Hwy. 22, Suite 2000
Bridgewater, NJ 08807
(908) 725-5445

**Walt Maximuck**
President
Solution Dynamics Inc.
981 US Hwy. 22, Suite 2000
Bridgewater, NJ 08807
(908) 725-5445

# VIRTUAL LOADER

## A DEVELOPMENT ENVIRONMENT FOR LOAD PLANNING SYSTEMS

Solution Dynamics Inc. uses ILOG Solver as the basis for a class library used to develop load planning applications for logistics organizations. The product has several advantages over other approaches: the ability to quickly add customer-specific business requirements; the ability to satisfy conflicting objectives; and, advanced load graphics utilities.

## 1. INTRODUCTION

This paper briefly discusses the experience of Solution Dynamics Inc. (SDI) in building solutions to different load planning problems. The first part presents a load planning development environment called Virtual Loader. Virtual Loader was developed by SDI as a set of C++ components on top of ILOG Solver. The second part explains how Virtual Loader has been used to build a load planning module for Thomson Consumer Electronics (TCE).

## 2. VIRTUAL LOADER

Virtual Loader is designed to be a flexible environment for developing affordable solutions to load planning problems. It has to be flexible because every customer seems to have a different combination of operational constraints which need to be accommodated. The foundation on which Virtual Loader is developed is ILOG Solver. We chose Solver because the solution search space of the loading problem, although very large, consists of a finite number of solutions which can be reduced by constraint propagation. Also, most constraints are logical constraints, so we needed an environment that would make it easy to embed logical concepts.

Virtual Loader is essentially a set of C++ components for representing and solving the load planning problem. These software components consist of objects, constraints and algorithms which define loading objects, loading space and quality of load. Even more importantly, Virtual Loader is extendible, providing the ability to address new problems as the market evolves.

### *2.1. Components*

Virtual Loader is composed of the following software components:

- Loading Core

- Loading Engines
- Input/Output Interfaces
- Load Visualization Component.

### 2.1.1. Loading Core

The Loading Core is a set of C++ classes used to represent different loading environments and loading requests. The names of some base classes are self-explanatory: LoadCore, LoadInventory, LoadProblem, LoadSolution, WhereToLoad, WhatToLoad, LoadConstraint, LoadRequest, LoadEngine, LoadStrategy, etc. For example, WhatToLoad contains the physical and operational characteristics of different products to be loaded. WhereToLoad defines the physical and operational characteristics of the loading spaces. It contains a detailed definition of the available capacity of the load space at each point in the loading process. LoadConstraints may define different problem-level, truck-level or product-level loading requirements, e.g. the orientation (upright, on side, on end) of product in the load.

Most of the Loading Core classes are abstract: they are used only to define interfaces. A load planning application always contains one instance of the LoadCore class. This instance knows how to retrieve the information, both static and dynamic, which defines each loading problem. Different load engines with predefined or customized load algorithms are then used, as needed, to build load solutions. (It's important to note that the same loading constraint "demons" control the modification of the load core consistency. This is true whether modification is through the graphics editor or by a LoadEngine. Also, an intelligent graphical user interface which facilitates what-if analysis can be built using the LoadingCore as an object server representing the current state of the loading system.)

### 2.1.2. Loading Engines

The loading algorithms are actually methods of the class LoadEngine. They deal with the LoadCore objects using different LoadStrategies while satisfying currently active LoadConstraints. Different LoadEngines may deal with some specialized classes inherited from WhereToLoad and WhatToLoad classes. LoadTrailer, LoadPallet and LoadSurface are examples of derived classes.

One predefined load engine contains methods which redefine the configuration of the load space after loading objects have been placed in the load space; and, select the "best next" subset of the total available space to be loaded during the loading process. It provides information to the load space on the state of the next surface to be loaded. This can impact the selection of loading method and/or loading object(s) used during the next cycle.

A concrete load engine makes the determination as to which loading methods are used during each loading operation. The examples of such methods are:

1) building flat surfaces (packing) which maximizes the utilization of horizontal space;
2) building columns and walls (stacking) which maximizes the utilization of vertical space.

LoadEngine objects include sophisticated queuing methods for WhatToLoad objects. These queuing methods have significant impact on both the stability of the load and labor costs associated with loading and unloading operations. There are multiple levels of, potentially conflicting, sequencing priorities. The product queue is often reconfigured many times during the search process depending on the current state of the LoadCore.

The degree of optimization done at each level within the system is primarily constrained by customer priorities. For instance, the loading algorithm can generate a pattern that is too complex and labor intensive to load. Most customers want simple row and column type patterns. These patterns can result in a less dense load; but, they reduce labor costs.

Local optimization can also be overridden by other system components. The queuing optimization of WhatToLoad objects can be overridden by WhereToLoad if the space available on the "next best" surface will not support the "next best" product in the loading queue.

### 2.1.3. Input/Output Interface

The Interface layer provides the connection between customer data sets and the Loading Core. It is composed of a set of API's designed to read and write to a number of different file formats and memory. It has three major functions:

1) Access information which provides a physical description of the product which needs to be shipped, the quantity of each product type, a description of the space(s) available for loading and the operational constraints which must be satisfied,
2) Convert that information into a format which the Loading Core can process; and,
3) Convert the results generated by the Loading Core into formats useable by the Load Visualization Component and other systems.

We currently have the ability to accept input in the form of flat files with text delimited format and a variety of other standard file formats. We are also able to accept user defined problem specific formats through class derivation. We have interfaced to Warehouse Management (WMS), Traffic Management (TMS) and Order Entry Systems. Additionally, we are able to use data in a "loadfile" format. This is a file format which contains the information required to accurately define the characteristics of a load. Since the "loadfile" format, which is currently used by at least three software vendors, seems to have become a de-facto standard, SDI has built a set of API's which allow us to read, write and modify the "loadfile".

From the users standpoint, one of the most important features of a load planning system is the output. Solutions need to be provided in a format that is easily interpreted by various people (dispatchers, drivers, order pickers, etc.) who need the results in order to accomplish their tasks. In addition to being able to generate custom output formats used by WMS and TMS vendors, we currently present loading results in two standard forms: an ASCII flat text file; and, binary files in "loadfile" format. The binary "loadfiles" allow us to use the sophisticated editing and viewing capability of LoadGraphics by Quest Software.

#### 2.1.4. Load Visualization

The Load Visualization Component consists of viewers, editors and report generators. It converts the solutions generated by a Loading Engine into high quality graphic representations of the location of each unit loaded. These graphics can be used by:

1) Warehouse personnel during picking and loading operations;
2) Customer Service personnel to verify available capacity before promising shipment to the customer;
3) Transportation personnel to order the properly sized equipment from your carriers; and,
4) Your customers during put-away operations.

The Load Visualization Component provides the ability to build powerful editing and viewing tools customized to our customers requirements.

### *2.2. Load Planning Applications*

SDI built several load planing applications using Virtual Loader C++ components. Here we briefly describe some of them:

Pallets of Tiered Product. While loading layers of mixed product types across multiple pallets, the goal is to minimize the number of pallets required. The application honors physical constraints such as pallet height and weight restrictions as well as each product's load bearing capacity or crush factor. The system also handles multiple conflicting objectives such as optimal picking order and maximum space utilization.

Mixed Product Pallets. The task is to load products of mixed shapes and sizes on a pallet. Objective is to provide efficient space utilization while supporting efficient picking/ loading operations. The system provides efficient 3-dimensional packing while honoring customer stacking and palletization constraints.

Truck Loading. Maximize the utilization of space inside a truck while at the same time generating stacking patterns which are economical to load and which honor customer shipping constraints. The application handles concepts such as "keep similar product together", "do not split product type across trucks", "do not split purchase orders across trucks", "ship highest profit margin product first", etc.

Shipper Select. Select an appropriately sized shipper(s) (truck, pallet or box) to ensure the entire load can be shipped.

SDI uses a number of different application Design Patterns based on the type of loading application we are building. For instance, truck loading and layered pallet loading virtually always include the column building algorithm. Shipper Select applications all use modifications of a shipper selection pruning algorithm.

# 3. LOAD PLANING MODULE FOR THOMSON CONSUMER ELECTRONICS

## *3.1. The Situation*

TCE sells home entertainment and audio and communications equipment. They ship product internally to various sites and to a wide variety of customers across North America. Product destination can be anything from an internal distribution center to an individual store location or a retailers distribution center. A shipment may consist of anything from a couple of cases of product to "many" truckloads. A shipment is defined as a group of products which will all leave a single site bound for a single destination on the same day.

The TCE product line varies greatly in size and shape. It includes everything from projection TV's and home entertainment systems to CD players, telephones and the accessories that go with them. Some of this product is stored in palletized units, other product is simply stacked in columns in the warehouse. Another group of products is shrink wrapped into 2x3x6 bundles called pseudo-pallets. The wide variety of shapes and sizes makes it impossible to use any "standard cube" analysis which will consistently make efficient use of truck capacity across the many product mixes that they are required to ship. Additionally, TCE is being bombarded with a wide variety of shipping requirements from their customers. These requirements can be broken down into several groups.

Product prioritization requirements determine which products or groups of products will be "bumped" from the load if there is not enough room.

Palletization requirements deal with how product is unitized. Some customers want everything palletized; others want nothing palletized and, others don't seem to care. When the unitized bundles are stacked on trucks it is often possible to double stack those bundles. In some cases there must be a pallet under each bundle, in others there should be a pallet only under the bottom unit; and, in others there should be no pallets at all. Finally, in a few cases, the customers receive shipments of palletized product only in quantities which make up full pallet tiers.

Product integrity constraints deal with how the product is spread across a multiple shippers (pallets or trucks). These constraints include such things as do not split:

1) Orders across shippers;
2) Product types across shippers;
3) SKU's across shippers;
4) Order lines across shippers.

Product exclusivity constraints do not allow the mixing of certain types of products in the same shipper.

Stacking constraints define what products may and may not be stacked on top of each other.

Although TCE wants to support customer requirements whenever possible, they also have a need to run the truck loading operation as efficiently as possible. As such, they want to be able to load product in as large a bundle as possible. This minimizes the number of trips a fork truck has to make to the truck and the amount of hand loading required. They therefore want to be able to pull product out of the warehouse stacked to the full height of the truck whenever possible. If product is palletized, they want to keep it palletized.

In addition to all of the above, TCE has its own requirements regarding placement of product on a truck. Some of these requirements deal with the size and durability of the product, others deal with how well it "rides" in the truck and others deal with sales and marketing requirements.

### *3.2. The Problem*

The mainframe system which was being used to support warehouse and traffic operations, was no longer supporting the needs of the business. The variety in the product line has grown rapidly as has the complexity of customer shipping requirements. It was becoming impossible to be sure that promised shipments would fit on any given truck until the loading process started. Planning was becoming impossible. In order to ensure they would be able to ship the customer what was promised, TCE employees began using a larger and larger "safety buffer". The result was less than acceptable capacity utilization.

### *3.3. The Solution*

TCE needed a system which would tell them before the shipping orders went to the warehouse, how many and what sized trucks were needed to ship the order. TCE contacted SDI in December of 1995 to determine whether on not Virtual Loader was a viable solution.

### *3.4. The Implementation*

The Load Planning Module (LPM) was implemented as a subsystem of the Order Fulfillment Process at TCE. An Order Entry System, a Warehouse Management System, a Traffic Management System together generate a problem interface file (PIF) as input for LPM. The PIF is a flat file which contains the following information:

1) Quantity of product to be shipped;

2) Available trucks for shipment;
3) Product prioritization;
4) Palletization requirements;
5) Product integrity requirements; and,
6) Truck availability.

After reading the PIF, the LPM accesses a product file which contains the physical description of the product in length, width, height and weight. It also contains information about how the product is stored in the warehouse. This includes whether or not it is palletized, its palletized dimensions and how many units fit on a pallet and in one tier on a pallet. Finally, it contains constraints on the orientation in which each product can be stacked.

The system then starts to load product. First it loads unpalletized product onto pallets if required. It then loads palletized and unpalletized product onto trucks. The order of load is based on product prioritization constraints which are generated based on both customer and TCE requirements.

The system honors all constraints mentioned above. When there are conflicts, the system uses a prioritization scheme to promote and demote conflicting constraints so as to satisfy overall operational priorities.

If the system fills a truck or pallet and is unable to load all of a product or group of products which are identified with a “don’t split” tag, it backtracks and begins loading a different set of products. The “don’t split” group then has priority on the next pallet or truck.

When solutions have been generated, graphical representations of the loads are printed for use by warehouse personnel and the sequence of load is passed to the WMS to generate picking orders. The quantity and size of trucks required to ship the order is sent to the transportation organization to order the right equipment from the carriers.

## 4. CONCLUSION

The Load Planning Module being installed at Thomson Consumer Electronics Americas Division in July combines all of the above functionality. It is integrated with TCE Warehouse Management, Traffic Management and Order Entry Systems. SDI spent about two calendar months customizing Virtual Loader to fit TCE’s requirements. LPM runs on a UNIX server supporting a production environment; and, on PC’s in support of the Customer Service organization in a planning mode. Thomson expects to saving in excess 3% of shipping costs. The Solver based approach provided us the flexibility to address the customer’s needs in a cost effective manner and thereby win this business.